Université de Montréal

Apprentissage de circuits quantiques par descente de gradient classique

par

Aldo Lamarre

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

31 Juillet 2020

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Apprentissage de circuits quantiques par descente de gradient classique

présenté par

Aldo Lamarre

a été évalué par un jury composé des personnes suivantes :

Frédéric Dupuis
(président-rapporteur)

Gilles Brassard
(directeur de recherche)

Alain Tapp
(membre du jury)

Sommaire

Nous présentons un nouvel algorithme d'apprentissage de circuits quantiques basé sur la descente de gradient classique. Comme ce sujet unifie deux disciplines, nous expliquons les deux domaines aux gens de l'autre discipline. Conséquemment, nous débutons par une présentation du calcul quantique et des circuits quantiques pour les gens en apprentissage automatique suivi d'une présentation des algorithmes d'apprentissage automatique pour les gens en informatique quantique. Puis, pour motiver et mettre en contexte nos résultats, nous passons à une légère revue de littérature en apprentissage automatique quantique. Ensuite, nous présentons notre modèle, son algorithme, ses variantes et quelques résultats empiriques. Finalement, nous critiquons notre implémentation en montrant des extensions et des nouvelles approches possibles. Les résultats principaux se situent dans ces deux dernières parties, qui sont respectivement les chapitres 4 et 5 de ce mémoire. Le code de l'algorithme et des expériences que nous avons créé pour ce mémoire se trouve sur notre github à l'adresse suivante : `https://github.com/AldoLamarre/quantumcircuitlearning`.

Mots-clef : apprentissage automatique, informatique quantique, apprentissage automatique quantique, circuit quantique, descente de gradient, rétropropagation, apprentissage de circuit quantique, optimisation sur la variété de Stiefel.

Summary

We present a new learning algorithm for quantum circuits based on gradient descent. Since this subject unifies two areas of research, we explain each field for people working in the other domain. Consequently, we begin by introducing quantum computing and quantum circuits to machine learning specialists, followed by an introduction of machine learning to quantum computing specialists. To give context and motivate our results we then give a light literature review on quantum machine learning. After this, we present our model, its algorithms and its variants, then discuss our currently achieved empirical results. Finally, we criticize our models by giving extensions and future work directions. These last two parts are our main results. They can be found in chapter 4 and 5 respectively. Our code which helped obtain these results can be found on github at this link : `https://github.com/AldoLamarre/quantumcircuitlearning`.

Key words : machine learning, quantum computing, quantum machine learning, quantum circuits, gradient descent, backpropagation, quantum circuit learning, optimisation on the Stiefel manifold.

Table des matières

Liste des tableaux

Table des figures

Remerciements

Je remercie mes parents pour leur soutien extraordinaire durant toutes mes études. Je remercie aussi mon directeur de recherche Gilles Brassard pour cette occasion de travailler dans ce nouveau domaine. Je tiens à remercier Louis Salvail, Alain Tapp et tous les autres membres du LITQ pour leur aide. Pour notre longue discussion lors de sa visite à Montréal, je tiens aussi à remercier Nathan Weibe. Finalement, je remercie Frédéric Hamel pour ses excellents commentaires et critiques du code.

Pour des remerciements que je n'aurais jamais voulu avoir à faire, je remercie le Dr. Meskawi et toute l'équipe de l'hôpital Fleury pour une chirurgie rapide et bien réussei ainsi que le Dr. Tanguay pour le suivi. Apprendre que l'on a le cancer est une triste nouvelle et te met dans tous tes états. En plus de calmer mes esprits, ces médecins sont parvenus à trouver un bloc opératoire pour mon cas dans de brefs délais, et ce durant le confinement causé par la pandémie de la COVID-19.

Introduction

En mélangeant de l'apprentissage automatique et de l'informatique quantique, on obtient en bon jargon du *hype square*. Derrière notre titre « Apprentissage de circuits quantiques par descente de gradient classique » se cache une tentative d'évaluation de cette *hype*. Veuillez nous pardonner les mots anglais « *hype square* » car la traduction suggérée, battage médiatique au carré, n'a pas le même « ~~*punch*~~ » ! Non ! Ici, ici on va dire force de frappe. En premier lieu de notre titre on a « Apprentissage de circuits quantiques ». Bon nous allons apprendre des circuits quantiques. Nous espérons que le *hype* reste encore au carré. Surtout, cher lecteur, si vous ne connaissez pas ce qu'est un circuit quantique. Soyez manipulé par les *buzzwords*. En deuxième lieu par contre, « par descente de gradient classique » c'est là où se cache notre évaluation de cette fameuse *hype*.

Par « par descente de gradient » nous entendons l'algorithme d'entrainement en apprentissage profond classique, le principal responsable des succès et du *hype* de l'apprentissage automatique classique. On lui doit non seulement les résultats très médiatisés de *AlphaGo*, *AlphaZero* et moins médiatisés de *leelachess*, mais aussi les succès en reconnaissance d'image, de vidéo, la traduction automatique et plusieurs autres. En pensant s'éloigner du *hype* on y retourne. Par « classique », nous entendons que votre ordinateur dans votre sous-sol peut exécuter notre algorithme. Cela donne dans l'hyperbole, il possible qu'il doive l'exécuter pendant quelque temps en mois ou en années. Nous utilisons classique à l'opposé de quantique pour mentionner ce qui se rapporte aux ordinateurs traditionnels possédés par quasi tout le monde versus les ordinateurs quantiques auxquels bien peu de personnes ont accès à des versions très expérimentales.

Pourquoi voudrions-nous apprendre des circuits quantiques à l'aide d'un algorithme classique ? Pour répondre à cette question, mettons-nous dans le contexte du début de ce projet

de maitrise. Le domaine de l'apprentissage automatique quantique était axé sur l'accélération et l'adaptation au quantique d'algorithmes déjà connus [8]. Nous trouvions les gains quadratiques pour certains algorithmes raisonnables. Par contre, nous doutions des gains exponentiels offerts, car ils nécessitent une bonne quantité de conditions non triviales à remplir. Nous détaillons ces conditions dans le chapitre 3. Nous étions sceptiques face à la quantisation, c'est-à-dire l'adaptation au quantique d'algorithmes déjà connus. Nous pensions que développer un algorithme à partir du quantique serait une meilleure approche. Ces méthodes étant très théoriques, il y avait peu de résultats empiriques. Pour l'apprentissage automatique classique, nous trouvions que la variante de la descente de gradient utilisée par les réseaux récurrents unitaires serait adaptable à l'apprentissage de circuits quantiques.

Face à des méthodes limitées et un manque de résultat empirique, nous nous disons que nous pourrions entrainer des petits circuits quantiques classiquement et évaluer empiriquement leurs résultats. Les circuits doivent être petits, car pour les entrainer il faut les simuler, ce qui nécessite un espace exponentiel en mémoire. Le déploiement de plusieurs petits ordinateurs quantiques expérimentaux encourageait nos espoirs que dans un éventuel futur, un circuit une fois entrainé soit exécuté sur un ordinateur quantique. Nous avons réussi à faire pour ce mémoire la partie d'entrainement, malheureusement sans l'exécution sur un ordinateur quantique, pour des raisons d'accès et de taille de circuit.

C'est dans le chapitre 4 que nous présentons notre algorithme d'apprentissage de circuit quantique et discutons de ses résultats. C'est avec ces résultats empiriques obtenus que nous tenterons d'évaluer cette *hype*. Attention ! Il ne faut pas allez trop vite ! Allons-y étape par étape. En étape zéro, pour faire bon informaticien, mentionnons que nous tenons pour acquis que vous, notre lecteur, avez des connaissances en apprentissage automatique *ou* en informatique quantique. « Ou » non exclusif bien sûr ! Connaitre les deux est très bénéfique, mais pour faciliter l'accès aux contenus de ce mémoire nous faisons cette hypothèse.

C'est dans cette optique qu'en étape un, nous présentons le calcul quantique au chapitre 1 suivi de l'apprentissage automatique au chapitre 2. Par la suite, en étape 2, nous entrons dans les détails de notre mise en contexte d'il y a quelques paragraphes avec une revue de littérature de l'apprentissage automatique quantique au chapitre 3. Cette introduction devrait vous avoir donné toutes les bases pour comprendre notre modèle dans le chapitre 4, l'étape trois. En étape quatre, comme toute solution apporte de nouvelles questions, au

chapitre 5, nous proposerons plusieurs extensions possibles de notre projet et des travaux futurs reliés à l'apprentissage automatique quantique en général. Finalement, en étape 5, nous concluons ce mémoire.

Bon assez de placotage. Il est temps de commencer. Bonne lecture !

Chapitre 1

Calcul quantique

Nous introduisons les notions de calcul quantique. La pédagogie de cette section repose sur l'algèbre linéaire liée au calcul quantique. Nous utilisons cette approche pédagogique pour faire un pont commun entre l'apprentissage profond et les circuits quantiques. L'objectif est de permettre aux gens travaillant en apprentissage automatique de comprendre ce mémoire.

1.1. Introduction

Le calcul quantique se trouve présentement dans une grande évolution. Plusieurs machines de calcul quantiques ont été déployées dans les dernières années. En 2016, IBM déploya le premier ordinateur quantique accessible à tous sur le web, un processeur quantique de cinq qubits. Les qubits étant l'équivalent quantique du bit classique, nous les utilisons pour mesurer les tailles des processeurs quantiques. Depuis, la taille des ordinateurs quantiques a suivi une progression analogue à la loi de Moore des processeurs classiques, leurs nombres de qubits doublant approximativement à chaque année.

Aujourd'hui, il existe des processeurs quantiques de 49, 53 et 72 qubits. De plus, Rigetti, une entreprise dans la conception et fabrication de processeurs quantiques, planifie la sortie d'un processeur de 128 qubits. Les fournisseurs de *cloud computing* tel Microsoft et Amazon ont annoncé leur offre d'accès commerciale à des processeurs quantiques. Suite à l'article de suprématie quantique de Google, Scott Aaronson conclut que nous sommes maintenant dans les débuts de l'ère des tubes à vide des ordinateurs quantiques. En lien avec l'apprentissage automatique quantique, les circuits variationnels, un algorithme d'apprentissage automatique quantique que nous détaillons au chapitre 3, se trouve parmi les expériences couramment exécutées sur les processeurs quantiques.

Nous amorcerons nos explications du calcul quantiques par l'introduction des qubits et des portes quantiques avec une approche basée sur l'algèbre linéaire. Cette approche illustre la partie simulation quantique de l'algorithme d'apprentissage automatique quantique développé pour ce mémoire, principalement, l'utilisation de bibliothèques d'algèbre linéaire et d'apprentissage automatique dans son implémentation.

1.2. Qubits

L'information classique est représentée par un ou plusieurs bits. Un bit peut prendre les valeurs 0, 1 ou *vrai*, *faux*. Un qubit peut prendre des valeurs à la fois de 0 et de 1 dites en superposition quantique. Ceci est en contraste avec les circuits analogues qui prennent des valeurs *entre* 0 et 1.

Définition 1.2.1 (Qubits)**.** *Les qubits sont une unité d'information quantique analogue au bit classique. Nous noterons les qubits avec la notation de Dirac, ket* $|\rangle$ *et bra* $\langle|$.

La notation de Dirac permet de comprendre la partie algébrique des qubits. Les états $|0\rangle$ et $|1\rangle$ correspondent au 0 et 1 d'un bit classique et algébriquement aux vecteurs colonnes $\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ et $\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$. Les combinaisons linéaires de ces vecteurs correspondent au phénomène de superposition quantique.

Celles-ci forment donc d'autres états possibles :

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle\,,$$

où α et β sont des nombres complexes composantes d'un vecteur de norme unitaire :

$$\|\alpha\|^2 + \|\beta\|^2 = 1\,.$$

Algébriquement, les coefficients α et β correspondent à ceux de la combinaison linéaire $\alpha\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)+\beta\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ et $|\psi\rangle$ correspond au vecteur résultant de cette combinaison linéaire $\left(\begin{smallmatrix}\alpha\\\beta\end{smallmatrix}\right)$. Nous avons donc que les vecteurs $\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ et $\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ forment une base orthonormée et que les vecteurs normalisés engendrés par cette base donnent les superpositions quantiques d'un qubit.

Quantiquement, les coefficients α et β représentent l'amplitude de probabilité ou pour faire plus court amplitude. La norme au carré de α donne la probabilité de mesurer $|0\rangle$ et la norme au carré de β donne la probabilité de mesurer $|1\rangle$. C'est de là que vient la contrainte $\|\alpha\|^2 + \|\beta\|^2 = 1$, i.e. que la norme au carré d'une amplitude représente une probabilité.

Cette approche s'étend aux états ayant plusieurs qubits. Nous avons aussi algébriquement une opération pour combiner deux états quantiques en un seul système : le produit tensoriel.

Exemple 1.2.2 (États quantiques sur deux qubits)**.** *Les états* $|00\rangle$,$|01\rangle$,$|10\rangle$,$|11\rangle$*, forment une base orthonormée,*

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle,$$

où α *,* β*,* γ *et* δ *sont des nombres complexes composantes d'un vecteur de norme unitaire :*

$$\|\alpha\|^2 + \|\beta\|^2 + \|\gamma\|^2 + \|\delta\|^2 = 1\,.$$

Encore une fois, les états $|00\rangle$,$|01\rangle$,$|10\rangle$,$|11\rangle$ correspondent algébriquement aux vecteurs $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$,$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$,$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$,$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$ respectivement. Les combinaisons linéaires de ceux-ci engendrent les superpositions

$$\alpha\begin{pmatrix}1\\0\\0\\0\end{pmatrix} + \beta\begin{pmatrix}0\\1\\0\\0\end{pmatrix} + \gamma\begin{pmatrix}0\\0\\1\\0\end{pmatrix} + \delta\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \begin{pmatrix}\alpha\\\beta\\\gamma\\\delta\end{pmatrix}.$$

Nous avons maintenant une définition d'un état arbitraire sur deux qubits. Observons ce qu'il se passe lors que nous combinons deux états de un qubit à l'aide du produit tensoriel.

Exemple 1.2.3 (Produit tensoriel de deux qubits)**.** *Les états* $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ *et* $|\phi\rangle = \gamma|0\rangle + \delta|1\rangle$*, leur produit tensoriel noté* $|\psi\rangle \otimes |\phi\rangle$ *est*

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix} \otimes \begin{pmatrix}\gamma\\\delta\end{pmatrix} = \begin{pmatrix}\alpha\gamma\\\alpha\delta\\\beta\gamma\\\beta\delta\end{pmatrix}.$$

Dans les états possibles sur deux qubits, avons des états comme $\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|10\rangle$ et $\frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle$ qui sont le résultat de $(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) \otimes |0\rangle$ et $(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) \otimes (\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle)$ respectivement. Il y a aussi l'état $\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$ qui n'est pas le résultat d'un produit tensoriel. Ces qubits ne proviennent pas de systèmes indépendants, il y a une *intrication* entre eux.

Définition 1.2.4 (Intrication)**.** *Il existe des états valides qui ne sont pas le résultat d'un produit tensoriel, par exemple* $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$*, pour lequel il n'existe aucune factorisation possible. On dit de ces états qu'ils sont intriqués.*

Nous donnons maintenant une définition pour des états quantiques de taille arbitraire et une définition pour la composition de systèmes quantiques.

Définition 1.2.5 (État quantique)**.** *Soit $\{|\xi_i\rangle\}_i$, un ensemble de vecteurs formant une base orthonormée. Un état quantique arbitraire est de la forme*

$$|\psi\rangle = \sum_i \gamma_i |\xi_i\rangle \,,$$

avec

$$\sum_i \|\gamma_i\|^2 = 1 \,.$$

La norme au carré des amplitudes γ_i donne la probabilité d'obtenir $|\xi_i\rangle$ à la mesure.

Remarque 1.2.6 (Facteur de phase globale)**.** *Le facteur de phase globale est un nombre complexe de norme un pouvant multiplier toutes les amplitudes d'un état. Ce type de facteur est permis par les conditions sur la norme. Bien qu'ils sont numériquement différents, les états formés de cette façon sont indiscernables entre eux et de l'original. C'est-à-dire qu'il n'existe aucun moyen permis par la mécanique quantique pour les différencier.*

La notion d'états indiscernables est liée à plus que le simple facteur de phase globale. Nous la définissons plus loin dans ce chapitre.

Définition 1.2.7 (Produit tensoriel)**.** *Soit $|\psi\rangle = \sum_i \alpha_i |i\rangle$ et $|\phi\rangle = \sum_j \beta_j |j\rangle$, deux états quantiques, le produit tensoriel de $|\psi\rangle$ et $|\phi\rangle$, noté $|\psi\rangle \otimes |\phi\rangle$, est*

$$|\psi\rangle \otimes |\phi\rangle = \sum_i \sum_j \alpha_i \beta_j |i\rangle \otimes |j\rangle \,,$$

par convention nous abrégeons $|i\rangle \otimes |j\rangle$ par $|i\rangle|j\rangle$ ou $|ij\rangle$ ce qui donne

$$|\psi\rangle \otimes |\phi\rangle = \sum_i \sum_j \alpha_i \beta_j |ij\rangle \,.$$

1.3. Opérations sur les états quantiques

Maintenant que nous avons des définitions d'états et de systèmes quantiques, nous allons voir comment algébriquement effectuer des opérations sur ceux-ci. Les opérations que nous décrivons sont les portes quantiques et la mesure. Nous commençons par une notion de registre quantique. Ceux-ci nous permettent de décrire des opérations sur un sous-système quantique. Les registres quantiques contiennent un ou plusieurs qubits et sont notés par un indice correspondant à leur étiquette. Par exemple, avec l'état $|\psi\rangle_A \otimes |\phi\rangle_B$ que nous pouvons également écrire $|\psi\rangle_A |\phi\rangle_B$, nous avons l'état $|\psi\rangle$ au registre A et l'état $|\phi\rangle$ au registre B.

Exemple 1.3.1 (Registres quantiques)**.** *Les registres quantiques aident beaucoup lorsque nous avons des états intriqués comme* $|\Phi+\rangle_{AB} = \frac{1}{\sqrt{2}}|00\rangle_{AB} + \frac{1}{\sqrt{2}}|11\rangle_{AB}$*. Nous pouvons appliquer une porte ou effectuer une mesure sur les registres A, B ou AB. Cela permet d'être clair et précis lorsque nous décrivons les opérations.*

Nous allons tout d'abord définir les portes quantiques, puis voir quelques exemples

Définition 1.3.2 (porte quantique)**.** *Les portes quantiques effectuent une transformation linéaire sur les qubits. Nous représentons l'application d'une porte sur un état quantique par une matrice unitaire carrée.*

La multiplication de cette matrice par le vecteur correspondant au qubit donne le résultat de la porte appliquée sur le qubit.

Définition 1.3.3 (Matrice unitaire)**.** *U est une matrice unitaire ssi* $UU^\dagger = U^\dagger U = \mathbb{I}$.

Pour faciliter la compréhension, voici quelques exemples de portes fréquemment utilisées.

Exemple 1.3.4 (Hadamard)**.** *La porte d'Hadamard aussi appelé transformation d'Hadamard que nous notons H envoie les états* $|0\rangle$ *vers* $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle$ *et* $|1\rangle$ *vers* $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |-\rangle$*. La matrice unitaire* $H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$ *représente cette transformation.*

Algébriquement $H|0\rangle$ consiste faire :

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

et $H|1\rangle$:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{pmatrix} .$$

Exemple 1.3.5 (Négation)**.** *La porte de négation quantique, que nous notons N, équivaut à la négation classique. Elle envoie les états* $|0\rangle$ *vers* $|1\rangle$ *et* $|1\rangle$ *vers* $|0\rangle$*. La matrice unitaire* $N = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ *représente cette transformation.*

Algébriquement $N|0\rangle$ consiste faire :

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

et $N|1\rangle$:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} .$$

Exemple 1.3.6 (Changement de phase)**.** *Le changement de phase que nous notons P envoie les états $|0\rangle$ vers $|0\rangle$ et $|1\rangle$ vers $-|1\rangle$. La matrice unitaire $P = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ représente cette transformation.*

Algébriquement $P|0\rangle$ consiste faire :

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

et $P|1\rangle$:

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = - \begin{pmatrix} 0 \\ 1 \end{pmatrix} .$$

Ces portes agissent sur un seul qubit. Elles ne peuvent donc pas générer de l'intrication. Il nous faut donc des portes de plus grandes tailles pour obtenir toutes les opérations quantiques. Nous donnons quelques exemples de portes de plus grande dimension.

Exemple 1.3.7 (Ou exclusif quantique/négation contrôlée)**.** *Le ou exclusif quantique autrement appelé négation contrôlée, que nous notons $CNOT$, envoie les états $|10\rangle$ vers $|11\rangle$, $|11\rangle$ vers $|10\rangle$ et laisse identiques les états $|00\rangle$ et $|01\rangle$. La matrice unitaire $CNOT = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right)$ représente cette transformation.*

Algébriquement $CNOT(\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle) = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} .$$

Observons que la négation contrôlée permet de créer de l'intrication tel que l'état $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ de notre exemple, car $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle = (\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle)|0\rangle$ n'est pas intriqué. Avec les exemples de portes que nous avons donnés, nous ne sommes toujours pas quantiquement universel, car il existe encore des opérations quantiques que nous ne pouvons pas réaliser, ni même approximer avec précision arbitraire, par exemple la porte de Toffoli que nous définissons maintenant.

Exemple 1.3.8 (Toffoli)**.** *La porte de Toffoli envoie les états $|110\rangle$ vers $|111\rangle$, $|111\rangle$ vers $|110\rangle$ et laisse identiques les autres états. Nous représentons cette transformation par cette*

matrice unitaire :

$$\begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0 \end{pmatrix}.$$

Avec la porte de Toffoli et la porte d'Hadamard, nous obtenons un ensemble de portes quantiquement universel. La porte de Toffoli a aussi la propriété d'être universelle pour le calcul classique réversible. Conséquemment, nous pouvons transformer un circuit classique possiblement non réversible en circuit quantique par le remplacement de ses portes par des Toffolis. Toutes les opérations quantiques doivent être réversibles. Nous pouvons obtenir leur inverse par la transposée conjuguée complexe de leur matrice unitaire.

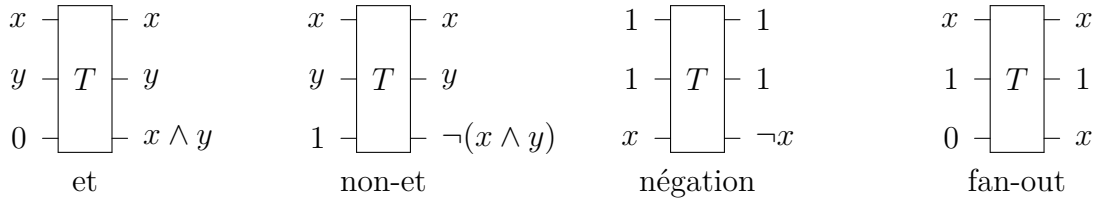


Figure 1.1. Implémentation de portes classiques à l'aide de la porte de Toffoli.

Tel qu'illustré à la figure 1.1 nous pouvons générer la porte non-et qui, avec la porte fan-out, est universelle classiquement ou la porte négation, la porte fan-out et la porte et, qui forment un ensemble classiquement universel.

Il n'est pas nécessaire d'utiliser une porte sur trois qubits pour créer un ensemble de portes quantiques universel À partir du $CNOT$, une porte de deux qubits, nous pouvons former un ensemble universel en permettant des portes unitaires arbitraires sur un qubit. Si nous exigeons un ensemble fini de portes, il manque jusqu'ici dans nos exemples certaines portes de un qubit pour obtenir l'universalité.

Exemple 1.3.9 (Changement de phase paramétré)**.** *Le changement de phase paramétré envoie les états* $|0\rangle$ *vers* $|0\rangle$ *et* $|1\rangle$ *vers* $e^{i\theta}|1\rangle$*. La matrice unitaire* $U_\theta = \left(\begin{smallmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{smallmatrix}\right)$ *représente cette transformation.*

Lorsque $\theta = \pi$ nous obtenons la porte P déjà introduite. Lorsque $\theta = \pi/4$ nous obtenons une porte appelée T. Avec $CNOT$, Hadamard et T, nous obtenons un ensemble universel fini de portes quantiques.

Théorème 1.3.10 (Théorème de Solovay–Kitaev)**.** *Un ensemble universel fini de petites portes suffit pour approximer une grande porte de taille arbitraire* n *avec précision* ϵ *en*

$O(4^n \log^c(\frac{1}{\epsilon}))$ *portes, où c est une constante appropriée. Pour des portes de même taille, un ensemble universel fini les approxime en* $O(\log^c(\frac{1}{\epsilon}))$ *portes.*

En pratique, nous sommes limités pour des raisons d'efficacité aux circuits de taille polynomiale avec $O(n^k \log^c(\frac{1}{\epsilon}))$ portes une fois approximées. Pour poursuivre nos explications des opérations, nous introduisons les circuits quantiques.

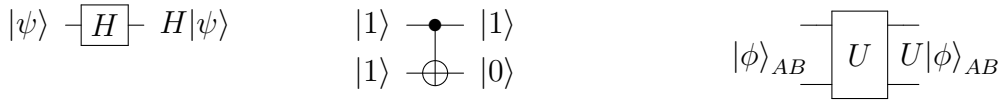
1.3.1. Circuits quantiques

Comme les circuits quantiques offrent un modèle très visuel, nous les montrerons par des exemples. En premier lieu à la figure 1.2, nous donnons en exemple la représentation en circuit de la porte d'Hadamard de la négation contrôlée et d'une porte arbitraire U sur deux qubits.

Hadamard CNOT Porte U sur deux qubits

Figure 1.2. Exemples de portes quantiques.

À la figure 1.3, nous montrons l'opération algébrique que ces circuits représentent. Pour la négation contrôlée, $-\bullet-$ représente le qubit de contrôle et $-\oplus-$ représente le qubit cible.



Hadamard CNOT Porte U sur deux qubits

Figure 1.3. Exemples de portes quantiques (suite).

Naturellement, les circuits peuvent combiner plusieurs portes. Nous donnons en exemple à la figure 1.4 le circuit créant les états de Bell.

Nous avons déjà mentionné les états de Bell $|\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ et $|\Psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$ à titre d'exemples d'états intriqués. Dans la figure 1.4 nous donnons les entrées nécessaires au circuit pour former les deux autres états de Bell $|\Phi^-\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$ et $|\Psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$.

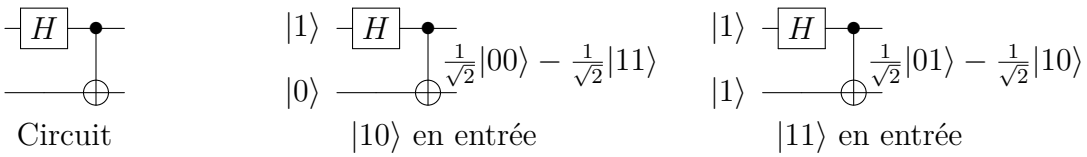


Figure 1.4. Circuit créant les états de Bell.

Le CNOT n'est pas la seule porte contrôlée. En fait, vous en avez déjà vu une deuxième, la porte de Toffoli est en fait un CNOT contrôlé donc une négation contrôlée contrôlée. « Contrôlée » n'est pas du « hype » on ne le met pas au carré. À la figure 1.5, nous présentons une porte contrôlée générique paramétrable avec n'importe quelle porte de un qubit. En plus, une porte très intéressante fait son apparition : la porte SWAP. Celle-ci permet d'interchanger ou « swapper » en bon chinois les qubits de deux registres. Comme ces portes n'ont pas été données en exemple auparavant, nous en profitons pour montrer leur matrice unitaire, sauf pour celle que nous avons déjà donnée : la porte de Toffoli, qui à ce sujet, tient à nous mentionner que sa matrice n'est pas grosse, mais bien enveloppée !

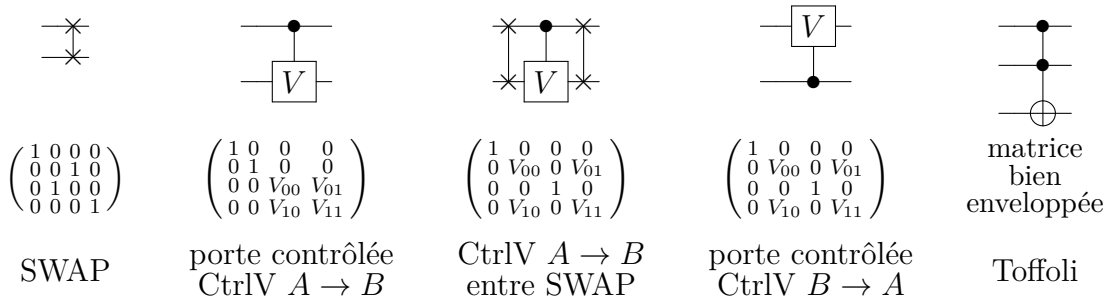


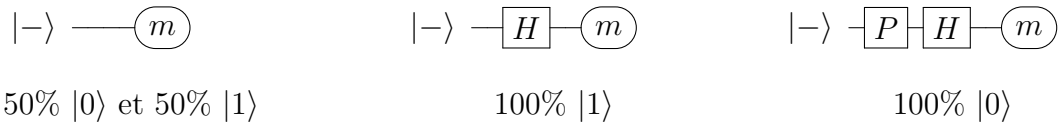
Figure 1.5. Exemple de SWAP et porte contrôlée avec leur matrice associée.

Au troisième circuit de la figure 1.5, le CtrlV du deuxième circuit utilise le SWAP du premier circuit pour inverser son qubit de contrôle avec son qubit de cible créant ainsi le CtrlV du quatrième circuit.

Dans la prochaine sous-section, nous utilisons les circuits quantiques pour montrer l'effet de la mesure sur les états quantiques.

1.3.2. Mesure

À la figure 1.6, nous montrons l'effet de la mesure sur le qubit $|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. La première mesure est faite dans la base de calcul, qui est constituée des vecteurs $|0\rangle$ et $|1\rangle$. Nous obtenons alors avec 50% de chance $|0\rangle$ comme résultat ou avec aussi 50% de chance $|1\rangle$ comme résultat. Pour une mesure dans une autre base, il est équivalent de faire une transformation unitaire comme changement de base puis une mesure dans la base de calcul. C'est ce que est fait pour la deuxième et troisième mesures.



50% $|0\rangle$ et 50% $|1\rangle$ 100% $|1\rangle$ 100% $|0\rangle$

Figure 1.6. Exemples de mesures sur un qubit et leur résultat.

Remarque 1.3.11 (Mesure projective)**.** *Les mesures données en exemple à la figure 1.6 sont des mesures projectives. On peut obtenir l'amplitude de probabilité de leurs résultats en faisant une projection entre l'état mesuré et la base de la mesure. Comme les vecteurs représentants les états quantiques sont de norme un la projection revient au produit scalaire.*

En calculant ces produits scalaires, nous confirmons les résultats des mesures de la figure 1.6. Pour la première mesure nous avons les amplitudes $\langle -|0\rangle = \frac{1}{\sqrt{2}}$ et $\langle -|1\rangle = \frac{-1}{\sqrt{2}}$. Ce qui nous donne comme probabilité $|\frac{1}{\sqrt{2}}|^2 = |\frac{-1}{\sqrt{2}}|^2 = \frac{1}{2}$ pour les deux résultats. Pour la deuxième mesure, elle peut être projetée directement dans la base de Hadamard ou on peut appliquer $H|-\rangle$ avant de projeter dans la base de calcul donnant les amplitudes $\langle -|+\rangle = \langle -|H|0\rangle = \langle 1|0\rangle = 0$ et $\langle -|-\rangle = \langle -|H|1\rangle = \langle 1|1\rangle = 1$. Similairement, *mutandis mutatis*, pour la troisième mesure qui donne les amplitudes $\langle -|-\rangle = \langle -|PH|0\rangle = \langle 0|0\rangle = 1$ et $\langle -|+\rangle = \langle -|PH|1\rangle = \langle 0|1\rangle = 0$.

De plus, à la troisième mesure de la figure 1.6, nous pouvons observer l'effet que le changement de phase $P = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ a dans la base de Hadamard qui est constituée des vecteurs $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ et $|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. Cette transformation est équivalente à la négation dans la base de calcul tel qu'illustré à la figure 1.7.

Figure 1.7. Effets du changement de phase P dans la base de Hadamard.

Nous pouvons vérifier mathématiquement l'équivalence de la figure 1.7 par ce produit matriciel :

$$HPH = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = N.$$

Pour des états à plus d'un qubit nous pouvons choisir de n'en mesurer qu'une partie. Nous appelons ce type de mesure une mesure partielle.

Remarque 1.3.12 (Mesure partielle)**.** *Fréquemment nous indiquons les registres quantiques sur lesquels elle s'applique. Par exemple, pour l'état* $|\psi\rangle_{ABC}$*, nous pourrions mesurer le registre A, B, C ou une combinaison de ceux-ci.*

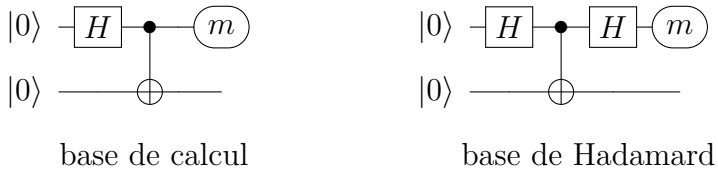


Figure 1.8. Mesures partielles de $|\Phi^+\rangle_{AB}$ sur le registre A.

À la figure 1.8, pour la première mesure de $|\Phi^+\rangle_{AB} = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ sur le registre A nous avons 50% de chance d'obtenir $|0\rangle$ ou 50% de chance d'obtenir $|1\rangle$. Si notre résultat est $|0\rangle$ alors le qubit sur le registre B est aussi $|0\rangle$ et si notre résultat est $|1\rangle$ alors le qubit du registre B est $|1\rangle$. La dépendance de l'état du registre B sur le résultat de la mesure du registre A est une propriété de l'intrication. Pour la deuxième mesure nous avons encore 50% de chance d'obtenir $|0\rangle$ ou 50% de chance d'obtenir $|1\rangle$. Par contre, si notre résultat est $|0\rangle$ alors le qubit sur le registre B est $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle$ et si notre résultat est $|1\rangle$ alors le qubit du registre B est $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |-\rangle$.

Remarque 1.3.13 (Mesure généralisée)**.** *Nous ne sommes pas limités aux mesures projectives. La mesure généralisée permet l'ajout de qubits ancillaires, qui sont des qubits additionnels permettant d'augmenter dans certains cas la précision de la mesure. On les implémente par un POVM, abréviation de positive-operator valued measure.*

Un exemple traditionnel de la mesure généralisée et des *POVM* est la distinction d'états quantiques. La distinction d'états ressemble fortement à la tache de classification en apprentissage automatique. La ressemblance ne s'arrête pas la. La mesure projective ressemble aux classificateurs linéaires et les *POVM* par leurs qubits ancillaires à l'astuce du noyau. Ces similarités motiveraient bien la présentation des *POVM*, cependant, ils ne sont pas nécessaires pour comprendre notre modèle d'apprentissage machine quantique. Pour bien les expliquer mathématiquement, il nous faudrait aller très en détail dans les notions d'états mixtes et de matrice densité. Toutefois, comme ces notions arrivent dans une partie de notre modèle, le choix de la fonction de perte, nous les montrons rapidement pour éviter de vous surprendre.

1.3.3. États mixtes

Exemple 1.3.14 (États mixtes)**.** *Dans la figure 1.8 de notre exemple de mesure partielle, si nous oublions le résultat après la mesure sur le registre A, l'état sur B à 50% de chance d'être* $|0\rangle$ *et 50% de chance d'être* $|1\rangle$ *pour la première mesure. Pour la deuxième mesure, l'état sur B a 50% de chance d'être* $|+\rangle$ *et 50% de chance d'être* $|-\rangle$*. Les états mixtes caractérisent cette possibilité d'être dans des états différents.*

Une des représentations des états mixtes est leur mélange statistique.

Définition 1.3.15 (Mélange statistique d'états quantique)**.** *Les mélanges statistiques d'états quantiques sont un ensemble de tuples d'état quantique* $|\psi\rangle_i$ *et de leur probabilité associée* λ_i *:* $\{(\lambda_i,|\psi_i\rangle)\}_{i=0}^{n-1}$.

Pour revenir à notre exemple, les mélanges statistiques de la figure 1.8 sont $S_0 = \{(\frac{1}{2},|0\rangle),(\frac{1}{2},|1\rangle)\}$ et $S_1 = \{(\frac{1}{2},|+\rangle),(\frac{1}{2},|-\rangle)\}$. L'autre représentation des états mixtes est leur matrice densité. La matrice densité d'un état pur $|\psi\rangle$ est formée par son produit externe avec lui-même $|\psi\rangle\langle\psi|$.

Définition 1.3.16 (Matrice densité)**.** *La matrice densité* ρ *de l'état au mélange statistique* $\{(\lambda_i,|\psi_i\rangle)\}_{i=0}^{n-1}$ *est la moyenne pondérée par les probabilités* λ_i *des matrices densité des états*

purs $|\psi_i\rangle\langle\psi_i|$ *:*

$$\rho = \sum_{i=0}^{n-1} \lambda_i |\psi_i\rangle\langle\psi_i| \,.$$

Les matrices densité sont souvent notées par les lettres grecques σ et ρ. Appliquer une porte U sur un état mixte se fait comme suit : $\sum_{i=0}^{n-1} \lambda_i U|\psi_i\rangle\langle\psi_i|U^\dagger = U\rho U^\dagger$. Donc, lorsque dans le chapitre 4 vous verrez des états quantiques sans $|\rangle$ vous saurez que nous parlons d'états mixtes. En bonus, nous profitons de la présentation des matrices densité pour définir formellement la notion d'états quantiques indiscernables.

Définition 1.3.17 (États indiscernables)**.** *Deux états quantiques quelconques sont indiscernables si et seulement si leur matrice densité sont égales.*

Encore une fois l'exemple de figure 1.8, oui, il est très polyvalent, nous avions les mélanges statistiques S_0 : $\{(\frac{1}{2},|0\rangle)(\frac{1}{2},|1\rangle)\}$ et S_1 : $\{(\frac{1}{2},|+\rangle)(\frac{1}{2},|-\rangle)\}$. Appelons la matrice densité du premier σ et celle du deuxième ρ.

$$\begin{aligned}
\sigma &= \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \\
&= \frac{1}{2}\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}1 & 0\end{pmatrix} + \frac{1}{2}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0 & 1\end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix}1 & 0\\0 & 0\end{pmatrix} + \frac{1}{2}\begin{pmatrix}0 & 0\\0 & 1\end{pmatrix} \\
&= \begin{pmatrix}\frac{1}{2} & 0\\0 & \frac{1}{2}\end{pmatrix} = \sigma
\end{aligned}
\qquad
\begin{aligned}
\rho &= \frac{1}{2}|+\rangle\langle +| + \frac{1}{2}|-\rangle\langle -| \\
&= \frac{1}{2}\begin{pmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\end{pmatrix} + \frac{1}{2}\begin{pmatrix}\frac{1}{\sqrt{2}}\\\frac{-1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix}\frac{1}{2} & \frac{1}{2}\\\frac{1}{2} & \frac{1}{2}\end{pmatrix} + \frac{1}{2}\begin{pmatrix}\frac{1}{2} & \frac{-1}{2}\\\frac{-1}{2} & \frac{1}{2}\end{pmatrix} \\
&= \begin{pmatrix}\frac{1}{2} & 0\\0 & \frac{1}{2}\end{pmatrix} = \rho
\end{aligned}$$

Leur matrice densité sont égales $\sigma = \rho$, les états sont donc indiscernables. En fait, non seulement l'Hardamard mais effectuer n'importe quelle transformation ou mesure sur le registre A ne va aucunement influencer la matrice densité du registre B. Ceci est lié à la propriété du non-signalement de la mécanique quantique. C'est-à-dire que si nous séparions le registre A et B en les positionnant à par exemple des années-lumière d'entre eux, il est impossible d'envoyer une information au propriétaire du registre B en faisant une quelconque opération sur le registre A et vice versa de B vers A.

Ceci dernier paragraphe ne prouve pas le non-signalement et ce n'est pas son objectif. Son but est de donner une certaine intuition sur pourquoi les phénomènes quantiques, incluant l'intrication, ne peuvent être utilisée pour communiquer plus vite que la vitesse de la lumière.

Avant de terminer, nous désirons présenter une borne très algorithmique sur l'information que nous pouvons extraire d'un état quantique : la borne de Holevo [15].

Remarque 1.3.18 (Borne de Holevo)**.** *La borne de Holevo dit que de tout état quantique de n qubits, nous pouvons seulement extraire n bits d'information*

Cette borne signifie que même s'il faut 2^n nombres complexes pour décrire les amplitudes d'un état quantique de n qubits, nous ne pouvons extraire que n bits de l'information contenue dans ceux-ci. Cette contrainte limite le calcul quantique. En effet, si un algorithme quantique qui ne fonctionne que sur n qubits doit produire au final plus que n bits d'information, les répétions de son exécution pourraient éliminer ses avantages.

Pour résumer ce chapitre, nous voulons souligner les points importants à la compréhension du chapitre 4. Premièrement, tel que mentionné, les *POVM* et les notions sur les états mixtes sont de faible importance pour notre travail. Simplement être familier avec la notation est suffisant. Deuxièmement, comme ce chapitre est destiné principalement aux gens du domaine de l'apprentissage automatique qui devraient être familiers avec l'algèbre linéaire des réseaux neuronaux, nous voulons donner l'essentiel pour une compréhension rapide du chapitre 4. Nous utilisons le quantique dans notre modèle comme des opérations d'algèbre linéaire, ligne fois colonne. C'est l'idée générale, les opérations quantiques sont simplement des opérations d'algèbre linéaire pour lesquelles vous êtes déjà familier, mais avec quelques contraintes supplémentaires non sur les opérations mais sur les objets : matrices, vecteurs. L'application d'une porte quantique peut être vue comme l'exécution d'un niveau complet de neurone d'un MLP linéaire. Il est certain que manœuvrer à travers les contraintes quantiques est très difficile, mais il est peu réaliste de vous en donner la capacité en un chapitre de mémoire. Nous souhaitons avoir suscité votre intérêt et que vous voudrez en savoir plus. Nous recommandons alors le manuel de référence de cette section *Quantum Computation and Quantum Information* [20] ou le manuel de notre directeur Gilles Brassard, que nous espérons voir publié un jour.

Chapitre 2

Apprentissage automatique classique

Nous présentons les bases en apprentissage classique et des modèles d'apprentissage profond. L'algèbre linéaire sera encore une fois l'approche pédagogique. En plus de faire un pont commun entre les domaines, notre objectif est d'introduire aux personnes en informatique quantique le langage du domaine et illustrer les similitudes algébriques entre certains modèles d'apprentissage et les circuits quantiques.

2.1. Apprentissage automatique

Pour l'apprentissage automatique, principalement l'apprentissage profond, la dernière décennie fut massive en développement. Leurs modèles et algorithmes passèrent de projet de recherche académique à se trouver omniprésent de nos jours. Les cyniques disent maintenant que le domaine se trouve dans une bulle économique et qu'un éventuel krash s'approche. Ceci est néanmoins très en contraste aux cyniques de jadis pour qui l'apprentissage profond ne fonctionnerait jamais. Nous souhaitons que la prochaine décennie soit autant marquante pour les ordinateurs quantiques.

Face à l'enthousiasme général accordé à toute l'intelligence artificielle depuis plusieurs années déjà, nous trouvons caduc de motiver leur étude. C'est pour cela que nous allons directement au vif du sujet. Nous commençons donc par une définition des algorithmes d'apprentissage.

Définition 2.1.1 (Algorithme d'apprentissage)**.** *On dit qu'un programme apprend de l'expérience E pour une classe de tâches T et une mesure de performance P, si sa performance aux tâches dans T tel que mesuré par P augmente avec l'expérience E. [19]*

Le reste de cette section servira à décrire différentes tâches T, des expériences E et des mesures de performance P, avec objectif de présenter aux informaticiens quantiques le vocabulaire du domaine. Naturellement, les exemples donnés ne forment pas une liste exhaustive de ce qui est réalisé en pratique.

2.1.1. Tâches

Nous débutons par des exemples de tâches. Les tâches sont ce que nous voulons que notre programme apprenne. Par exemple, si nous voulons un programme qui joue aux échecs alors sa tâche sera de jouer aux échecs. Nous décrivons celles-ci en termes de la sortie attendue de l'algorithme.

- Classification : Pour classifier, nous demandons à notre programme de spécifier à quelle classe parmi k son entrée appartient. Typiquement, le programme sortira l'étiquette de la classe appropriée ou une distribution de probabilité sur les k classes.
- Regression : Pour cette tâche, l'algorithme prédit une valeur numérique associée à son entrée.
- Estimation de densité de probabilité : l'algorithme apprend une fonction de densité sur l'espace des données.

Cette liste n'est pas exhaustive. Ces exemples de tâches ont été donnés car elles apparaissent fréquemment dans les cours d'apprentissage automatique.

2.1.2. Mesures de performance

La mesure de performance P dépend de la tâche donnée à l'algorithme. Elles peuvent être différentes de la fonction de perte ou de coût que nous définissons à l'entrainement. Voici deux mesures de performance traditionnelles : la précision et le taux d'erreur, ce dernier étant simplement un moins le premier.

Définition 2.1.2 (Précision)**.** *La précision est le taux de succès de notre algorithme. C'est-à-dire le ratio de bonnes réponses.*

Nous pouvons aussi obtenir la précision en mesurant le taux d'erreur de notre algorithme. Comme ce que nous voulons mesurer est une erreur de généralisation, nous devons estimer

sa performance sur des nouvelles données. C'est pour cela que nous l'évaluons avec un ensemble de données n'ayant pas été utilisé à l'entrainement. Les fonctions de perte ou de coût que nous utilisons à l'entrainement en général différent des mesures ou métriques évaluant notre modèle. Celles-ci possèdent dans la plupart des cas des propriétés mathématiques, de statistiques ou de théorie d'information facilitant l'apprentissage tel qu'être continue, différentiable, monotone, être un estimateur statistique, maximisant une information mutuelle, etc.

2.1.3. Type d'apprentissage

Par type d'apprentissage, nous entendons l'expérience utilisée pour apprendre par l'algorithme. Règle générale, nous la découpons en trois catégories, apprentissage par renforcement, apprentissage supervisé et apprentissage non supervisé. Ceux-ci diffèrent en général sur la forme et le traitement de l'ensemble de données utilisé pour les entrainer.

- Apprentissage non supervisé : le programme tente d'apprendre des propriétés sur la structure de l'ensemble des données. Par exemple, une estimation de densité de probabilité sur les données ou pour du *clustering* pour lequel nous séparons les données en groupes d'exemples similaires. Les fonctions de perte de ces algorithmes se basent sur des propriétés statistiques des données.
- Apprentissage supervisé : Nous associons une étiquette ou une cible à chaque exemple de l'ensemble de données. Par exemple, l'ensemble de données *mnist*, pour lequel chaque image a été associée au chiffre entre 0 et 9 écrit sur celle-ci. L'algorithme apprend à classifier ces images dans ces dix classes de chiffres. Les fonctions de perte et la mesure de performance vont dépendre des étiquettes ou cibles ayant été associées aux exemples.
- Apprentissage par renforcement : Ces algorithmes interagissent avec leur environnent. Les fonctions de perte donnent des punitions ou des récompenses selon les actions prises par l'algorithme. La conception de ces fonctions et l'évaluation de ces modèles présentent des problèmes non triviaux.

Pour ce mémoire, nous nous concentrons sur l'apprentissage supervisé et la classification, car c'est de cette façon que nous allons évaluer notre modèle.

2.1.4. Capacité d'apprentissage

La capacité d'apprentissage est l'espace de fonctions atteignable par notre modèle. Un modèle ayant une plus grande capacité peut modéliser des fonctions plus précises. Contrairement au processus d'optimisation, obtenir une meilleure précision sur nos données d'entrainement peut désavantager un algorithme d'apprentissage. En effet, ce que nous minimisons est une erreur de généralisation. Pour entrainer notre algorithme, nous minimisons une fonction de coût ou de perte sur un ensemble de données d'entrainement. En évaluant la performance sur ces données nous obtenons l'erreur d'entrainement. En évaluant la performance sur un ensemble de donnée de test, des nouvelles données n'ayant pas été utilisées pour entrainer notre algorithme, nous obtenons un estimé de l'erreur de généralisation autrement appelée l'erreur de test.

Définition 2.1.3 (Ensemble d'entrainement)**.** *Ensemble de données sur lequel nous entrainons notre modèle.*

Définition 2.1.4 (Ensemble de validation)**.** *Ensemble de données sur lequel nous validons nos modèles. Il sert à comparer plusieurs modèles.*

Définition 2.1.5 (Ensemble de test)**.** *Ensemble de données sur lequel nous testons notre modèle final. Il sert à estimer la performance du modèle sélectionné.*

Les algorithmes avec grande capacité tendent à apprendre les données d'entrainement par cœur et conséquemment à avoir une faible erreur d'entrainement et une plus grande erreur de test. Au contraire, les algorithmes avec faible capacité sont incapables d'apprendre complètement la structure derrière les données et ont donc une plus grande erreur d'entrainement. Nous appelons ces phénomènes le sur-apprentissage et le sous-apprentissage, respectivement. Notons que des facteurs autres que la capacité peuvent les influencer. En pratique, pour éviter de se retrouver dans ces cas, nous créons un nouvel ensemble de données que nous appelons ensemble de validation sur lequel nous pouvons évaluer la performance de nos modèles sans biaiser nos données de test.

Définition 2.1.6 (Sur-apprentissage)**.** *Cas où le modèle obtient de très bonnes performances sur les données d'entrainement, mais de moins bonnes performances sur de nouvelles données.*

Définition 2.1.7 (Sous-apprentissage)**.** *Cas où le modèle obtient des mauvaises performances sur les données d'entrainement.*

En général, nous évaluons plusieurs modèles ou plusieurs variantes d'un même modèle. Si nous utilisons l'ensemble de test pour les comparer nous risquons de sur-apprendre sur celui-ci et biaiser notre estimation de notre erreur de généralisation. L'ensemble de validation nous permet de vérifier ou plus précisément valider nos performances sans introduire de biais dans nos résultants. Par contre, il faut faire attention, car si nous le sur-utilisions, il y aurait un risque de sur-apprentissage, risque que nous allons seulement découvrir à la fin de notre apprentissage au moment où nous évaluons sur nos données de test. C'est pour cela qu'il existe des techniques, telle la validation croisée, permettant de pallier ce problème. Dans ce mémoire, nous ne les monterons pas, car elles n'ont pas été utilisées. Pour plus d'information à leur sujet, nous avisons de consulter un manuel d'apprentissage automatique.

Définition 2.1.8 (Régularisation)**.** *Modification apportée à un algorithme d'apprentissage dans le but de minimiser l'erreur de généralisation sans minimiser l'erreur d'entrainement*

L'utilisation d'un ensemble de validation pour vérifier l'apprentissage de notre algorithme est une approche coûteuse pour contrôler la capacité de nos modèles. Elle nécessite l'entrainement de plusieurs configurations de modèles différentes, pour par la suite rejeter ceux en sous-apprentissage et sur-apprentissage. C'est une façon brute de couper l'espace de fonction accessible au modèle pour limiter le sur-apprentissage. Nous pouvons alternativement réduire le sur-apprentissage sans limiter notre espace de fonction et risquer de se retrouver en sous-apprentissage. En pénalisant les modèles ayant une plus grande capacité à l'entrainement, nous conservons notre espace de fonctions mais pour des performances similaires nous préférons les modèles plus simples. Par le principe du rasoir d'Occam, nous conjecturons que les modèles plus simples performeront mieux sur des nouvelles données. Nous appelons ce genre de technique de la régularisation.

2.1.5. Hyperparamètres

À l'instar des statistiques, nous avons aussi de méthodes paramétriques et non paramétriques. De façon évidente, les méthodes paramétriques possèdent des paramètres que le modèle va apprendre et les méthodes non paramétrique n'en possèdent pas. Par contre, nous pourrions considérer l'ensemble d'entrainement comme les paramètres des méthodes non

paramétriques, car leurs prédictions dépendent directement de cet ensemble. Nous allons utiliser cette simplification pour faciliter nos exemples. Cette approche considère le nombre de paramètres comme une approximation en ressources computationnelles nécessaires à notre modèle. Alors, tous les modèles possèdent des paramètres, i.e. des valeurs qu'il va apprendre. Ils possèdent aussi des valeurs qui se seront pas apprises que nous appelons hyperparamètres. Ceux-ci sont très importants, car ils contrôlent la capacité d'apprentissage et l'efficacité en ressources informatiques de notre modèle.

Exemple 2.1.9 (Classificateur à voisins les plus proches)**.** *Comme exemple, nous utilisons le classificateur à plus proche voisin. C'est une méthode non-paramétrique, mais nous y allons de notre simplification en considérant son ensemble d'entrainement comme ses paramètres. Le classificateur à plus proche voisin est un apprentissage supervisé qui regarde à l'aide d'une notion de distance D, un nombre k de voisins les plus proches et fait un vote sur leur étiquette.*

Le nombre k de voisins considérés et la distance D choisie ne sont pas appris. Ce sont ici des hyperparamètres. Dépendant de leur choix ils vont affecter les performances du modèle. Évidemment, l'algorithme va devoir calculer la distance et le nombre de voisins affecte légèrement le coût de la recherche. Ces coûts vont dépendre de la taille de l'ensemble d'entrainement.

2.1.6. Maximum de vraisemblance

Nous avons mentionné fréquemment l'évaluation de la performance de l'algorithme sur des nouvelles données pour estimer l'erreur de généralisation. Par le principe du maximum de vraisemblance, nous trouvons un bon estimateur au sens statistique de cette erreur. Nous n'entrons pas dans les détails des estimateurs, car ils relèvent plus du domaine de la statistique. Nous définissons le maximum de vraisemblance car il est au cœur des fonctions de perte utilisées à l'entrainement.

Considérerons un ensemble de données $X = \{x_0,x_1,\ldots,x_{n-1}\}$ iid provenant de la distribution $D(x)$ de la tâche que nous voulons apprendre et M_θ un modèle d'apprentissage automatique paramétré par θ. Soit $p(x|\theta)$ la probabilité d'obtenir x en sortie de notre modèle sachant ses paramètres θ. L'estimateur de θ par maximum de vraisemblance est :

$$\theta_{MV} = \arg\max_\theta p(X|\theta).$$

Comme X est iid :

$$\theta_{MV} = \arg\max_{\theta} \prod_{i=0}^{n-1} p(x_i|\theta).$$

Puisqu'un produit de probabilité a des défauts calculatoires tel qu'être susceptible aux erreurs numériques, nous utilisons la propriété monotone croissante de la fonction logarithmique pour la transformer en somme.

$$\theta_{LV} = \arg\max_{\theta} \sum_{i=0}^{n-1} \log p(x_i|\theta).$$

Pour des raisons numériques, nous préférons minimiser une fonction. Donc maximiser revient à minimiser sa négation.

$$\theta_{LV} = \arg\min_{\theta} \sum_{i=0}^{n-1} -\log p(x_i|\theta).$$

Nous pouvons l'interpréter comme maximiser la log-probabilité ou minimiser l'entropie croisée. Nous définissons cette dernière à la section suivante (définition 2.2.8).

Suite à ce survol des principes de base de l'apprentissage automatique, nous présentons des modèles d'apprentissage automatique.

2.2. Réseaux neuronaux

Nous montrons les réseaux neuronaux non seulement en raison de leur performance. Bien qu'ils soient un excellent modèle d'apprentissage automatique et notre compétiteur direct dans la plupart des tâches sur lesquelles nous évaluerions un circuit quantique, nous avons des raisons additionnelles de les présenter. Pour concevoir notre algorithme d'apprentissage de circuit, ils furent notre inspiration principale. Premièrement, nous avons modifié leur processus d'entrainement, la descente de gradient, pour apprendre des circuits quantiques. Par la suite, comme les deux modèles s'entrainent par une méthode commune, nous pouvons combiner les deux architectures, créant ainsi un modèle hybride, pour lequel un réseaux neuronal et un circuit quantique communiquent et apprennent en coopérations.

Pour débuter, nous définissons le neurone, l'unité de base de réseaux neuronaux. Puis nous présentons une architecture de réseaux neuronaux : les réseaux à propagation directe. Ensuite nous donnons quelques exemples de fonctions d'activation. Finalement, nous montrons des

fonctions de coût utilisées à l'entrainement. La notation traditionnelle de l'apprentissage automatique met les vecteurs (colonne) en ***gras***, les matrices en ***GRAS*** majuscule et les scalaires en *italiques*.

Définition 2.2.1 (Neurone)**.** *Un neurone est défini par un vecteur de poids w, un biais b et une fonction d'activation* act*. Il effectue l'opération mathématique suivante :*

$$y = \text{act}(\mathbf{w}^T\mathbf{x} + b)$$

En général, la fonction d'activation est non-linéaire. Un réseau neuronal limité aux activations linéaires reste linéaire, peu importe sa profondeur. Obtenir leur pleine capacité d'apprentissage nécessite l'utilisation des non-linéarités.

Définition 2.2.2 (Réseaux neuronaux à propagation directe)**.** *Un réseau neuronal à propagation directe est un réseau ayant plusieurs neurones par niveau, dont chaque neurone d'un niveau est connecté aux neurones du niveau suivant. Un niveau effectue cette opération mathématique :*

$$\boldsymbol{y} = \text{act}(\boldsymbol{W}^T\mathbf{x} + \mathbf{b})$$

On combine plusieurs de ces niveaux en couches successives pour obtenir un réseau neuronal. Les $\text{act}_f \dots \text{act}_0$ correspondent aux fonctions d'activation des différents niveaux.

$$\mathbf{y} = \text{act}_f(\mathbf{W}_f^T\text{act}_{f-1}(\mathbf{W}_{f-1}^T\text{act}_{f-2}(\dots\text{act}_0(\mathbf{W}_0^T\mathbf{x} + \mathbf{b}_0)\dots) + \mathbf{b}_{f-1}) + \mathbf{b}_f)$$

La profondeur d'un réseau provient de ce nombre de niveaux. Les niveaux n'interagissant pas avec l'environnent, c'est-à-dire ceux qui ne reçoivent pas d'entrées ou n'émettent pas de sortie sont appelés niveaux cachés ou couches cachées. Nous appelons les réseaux ayant plusieurs niveaux des réseaux profonds.

2.2.1. Fonction d'activation

Il est important de considérer la fonction d'activation d'un réseau neuronal. Une non-linéarité ne suffit pas pour obtenir un réseau compétent. Il faut assurer la propagation du gradient sur toute la profondeur du réseau pour réussir à l'entrainer. Les gradients ont tendance à tendre vers l'infini, i.e. explosion du gradient ou vers zéro, i.e. disparition du gradient plus un réseau est profond. Voici quelques exemples de fonctions d'activation pour

lesquelles nous présentons leur utilisation courante en pratique et mentionnons leur capacité de propagation des gradients.

Définition 2.2.3 (relu)**.**

$$\text{relu}(x) = \max(0,x)$$

La fonction *relu (rectified linear unit)* est fréquemment utilisée, car elle limite en pratique l'explosion et la disparition du gradient et permet des réseaux plus profonds. Même si sa dérivée en $x = 0$ n'existe pas, en pratique cette valeur arrive peu et est souvent le cas d'un arrondi ou d'une faible précision numérique. Les implémentations fixent alors la dérivée en $x = 0$ à 0 et ignorent le problème. Pour des cas précis où cela pourrait causer problème, il existe des généralisations de *relu* éliminant ce défaut. Nous ne les présenterons pas dans ce mémoire.

Définition 2.2.4 (sigmoïde)**.**

$$\sigma(x) = \frac{1}{1 - e^{-x}}$$

La sigmoïde est utilisée dans le niveau de sortie lors d'une classification binaire. Contrairement au softmax défini ci-dessous, qui est sa généralisation multi-classe, elle fut couramment utilisée dans les niveaux cachés des réseaux neuronaux. En pratique, de nos jours, elle est limitée aux implémentations spécialisées ayant été remplacée par le relu qui limite l'explosion du gradient.

Définition 2.2.5 (tanh)**.**

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

La fonction tangente hyperbolique est liée à la sigmoïde par la relation mathématique $\tanh(x) = 2\sigma(2x) - 1$. Cette relation lui donne des propriétés mathématiques similaires à la sigmoïde. Ceci a conduit à son remplacement par la fonction relu. Son avantage face à la sigmoïde est sa similarité à l'identité aux valeurs proche de zéro. Pourvu que la valeur absolue des préactivations reste proche de zéro, un réseau avec ces fonctions ressemble à un réseau linéaire, ce qui facilite l'entrainement. Au final, elle est comme la sigmoïde reléguée aux implémentations spécialisées.

Définition 2.2.6 (Softmax)**.**

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Le *softmax* est utilisé dans le niveau de sortie lors de la classification multiclasse. Il généralise la sigmoïde qui correspond à son cas binaire i.e. où nous avons seulement deux classes.

Nous passons aux fonctions de coût sans expliciter les propriétés mathématiques des diverses fonctions d'activation, principalement celles reliées aux problèmes de propagation du gradient, dans le but rester dans le sujet. Nous allons mentionner les problèmes de propagation du gradient lorsqu'ils reviennent, mais pour notre sujet nous n'avons pas eu à les résoudre à l'aide de fonctions d'activation.

2.2.2. Fonctions de coût

Nous entrainons les réseaux neuronaux par la minimisation d'une fonction de coût ou perte, termes que nous utilisons interchangeablement. Choisir une fonction de perte appropriée aide l'apprentissage. Une bonne fonction de coût devrait faciliter la tâche d'optimisation et donc permettre d'apprendre plus rapidement sur les données d'entrainement. Aussi elles doivent être un bon estimateur de l'erreur de généralisation. Ainsi elle découlent souvent du principe du maximum de vraisemblance.

Définition 2.2.7 (Erreur quadratique moyenne)**.**

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1} |x_i - y_i|^2$$

L'erreur quadratique est utilisée surtout par facilité mathématique. Selon la tâche à résoudre, elle est un bon estimateur du maximum de vraisemblance. Pour ces raisons elle est souvent la première fonction de perte utilisée pour de nouvelles approches lorsque des fonctions plus spécialisées n'ont pas encore été développées.

Définition 2.2.8 (Entropie croisée)**.**

$$H(x,y) = \sum_i -y_i \ln(x_i)$$

L'entropie croisée est une fonction de perte associée aux *softmax* et aux problèmes de classification en général. En pratique, nous utilisons l'entropie croisée pour entrainer les réseaux neuronaux ayant des sorties *softmax*. En appliquant le logarithme de vraisemblance sur le *softmax* nous la retrouvons.

2.3. Rétropropagation

Pour minimiser ces fonctions de coût, nous utilisons la descente de gradient. Nous consacrons une section entière à la descente de gradient dû à son importance capitale dans notre apprentissage de circuit quantique. Plus précisément, nous expliquons la rétropropagation, qui est une implémentation efficace de la descente de gradient. Cet algorithme calcule le gradient de la fonction de perte en fonction des poids du modèle selon la règle de la dérivée en chaine. Il évite le recalcul de valeurs intermédiaires.

C'est son efficacité qui donne aux réseaux neuronaux leur puissance d'apprentissage. C'est sans coïncidence que ce fut lorsque notre puissance de calcul est devenue assez élevée pour exécuter un réseau profond en temps raisonnable que l'apprentissage profond explosa. L'utilisation de la mémoire, le temps de calcul et la parallélisation permettent de mettre à jour un modèle dans environ le même ordre de temps qu'il faut pour exécuter une propagation avant. Propagation avant signifie ici donner une entrée à notre réseau et obtenir une sortie. Propagation arrière ou rétropropagation signifie de calculer les gradients des paramètres puis mettre à jours notre modèle selon les résultats d'une propagation avant.

2.3.1. Descente de gradient

Dans une vision de haut niveau la descente de gradient est une procédure de minimisation de fonctions. Nous nous déplaçons sur la courbe de la fonction dans la direction de ses gradients pour nous diriger vers son minimum.

Algorithme 1 : Descente de gradient (esquisse)

$F(x;\theta)$: Fonction paramétrée par θ que nous voulons minimiser.

$\nabla_\theta F(x;\theta) = \frac{\partial F(x;\theta)}{\partial \theta}$: Gradient de $F(x;\theta)$ par rapport à θ.

E : Nombre d'époques

λ : Le taux d'apprentissage.

for $i = 0$ **to** $E - 1$ **do**
    $\theta_{i+1} = \theta_i - \lambda \nabla_{\theta_i} F(x;\theta_i)$

L'algorithme 1 est une version simpliste de la descente de gradient pour l'apprentissage automatique. Ici, nous arrêtons nos mises à jour après E époques, ce qui correspond au nombre d'itérations de l'algorithme. Parfois, nous bénéficierons de faire un arrêt plus tôt disons lorsque $|F(x;\theta)| < \epsilon$. De plus, nous avons omis de préciser ce qu'est la variable x. Est-ce toutes nos données, une seule donnée, un sous-ensemble de nos données ou un bonhomme

de neige ? Surprise, x n'est pas un bonhomme de neige ! x représente un ensemble de nos données. Lorsque x contient toutes nos données, nous parlons de descente de gradient et notre algorithme revient à l'algorithme 1 avec une précision importante !

Cette précision importante est que nous minimisons alors le risque empirique de nos données c'est-à-dire la moyenne de la fonction de perte $\forall x \in X$. Donc notre objectif à minimiser et son gradient deviennent :

$$F(X;\theta) = \frac{1}{n}\sum_{j=0}^{n-1} F(x_j;\theta) \text{ et } \nabla_\theta F(X;\theta) = \frac{1}{n}\sum_{j=0}^{n-1} \nabla_\theta F(x_j;\theta)$$

résultant en cet algorithme duquel nous enlevons la définition des variables se trouvant déjà dans l'algorithme 1.

Algorithme 2 : Descente de gradient

$x \in X$: Les petits x sont les éléments de notre ensemble grand X de données de taille n .

for $i = 0$ **to** $E - 1$ **do**

  $\theta_{i+1} = \theta_i - \lambda \frac{1}{n} \sum_j^n \nabla_\theta F(x_j;\theta)$

Lorsque x contient un seul exemple, nous parlons de descente de gradient stochastique. Pour l'algorithme suivant et les prochains de cette sous-section, nous continuons à ne pas spécifier les variables déjà définies dans les précédents.

Algorithme 3 : Descente de gradient stochastique

for $k = 0$ **to** $E - 1$ **do**

  $\hat{\theta}_0 = \theta_k$

  for $i = 0$ **to** $|X| - 1$ **do**

    $\hat{\theta}_{i+1} = \hat{\theta}_i - \lambda \nabla_{\hat{\theta}_i} F(x_i;\hat{\theta}_i)$

  $\theta_{k+1} = \hat{\theta}_{|X|}$

Finalement, lorsque x contient un sous-ensemble de nos données, nous parlons de descente de gradient par *batch* ou *minibatch*. Dans ce cas nous minimisons le risque empirique sur une *batch*. Pour une *batch* B de taille b et $\forall x_j \in B$, notre objectif à minimiser et son gradient deviennent alors la moyenne des gradients de la fonction de perte des exemples de la *minibatch* :

$$F(X;\theta) = \frac{1}{b}\sum_{j=0}^{b-1} F(x_j;\theta) \text{ et } \nabla_\theta F(X;\theta) = \frac{1}{b}\sum_{j=0}^{b-1} \nabla_\theta F(x_j;\theta)$$

Algorithme 4 : Descente de gradient par *batch* ou *minibatch*

b : Taille des sous ensembles de notre ensemble de données X dit *batch*.

$n = \left\lceil \frac{|X|}{b} \right\rceil$ (nombre de *batch*)

$r = |X| - b(n-1)$ (taille de la dernière *batch*)

for $k = 0$ **to** $E - 1$ **do**

$\hat{\theta}_0 = \theta_k$

for $i = 0$ **to** $n - 2$ **do**

$\hat{\theta}_{i+1} = \hat{\theta}_i - \lambda \frac{1}{b} \sum_{j=0}^{b-1} \nabla_{\hat{\theta}_i} F(x_{j+ib}; \hat{\theta}_i)$

$\theta_{k+1} = \hat{\theta}_{n-1} - \lambda \frac{1}{r} \sum_{j=0}^{r-1} \nabla_{\hat{\theta}_{n-1}} F(x_{j+(n-1)b}; \hat{\theta}_{n-1})$

Par abus de langage, la descente de gradient réfère indifféremment à toutes ces variantes. Nous venons de montrer une vision haut niveau de la descente de gradient. Lorsque nous rentrons dans le détail dans une vision plus bas niveau, c'est là que la rétropogation entre en jeux. Celle-ci va s'occuper du calcul de $\nabla_\theta F(x_j; \theta)$, plus précisément puisque son algorithme bénéficie du parallélisme, il s'occupe de calculer $\frac{1}{b} \sum_{j=0}^{b-1} \nabla_\theta F(x_j; \theta)$ efficacement.

2.3.2. Graphe de calcul

Pour décrire l'algorithme de rétropogation, nous définissons les graphes de calcul, qui est un langage permettant de visualiser les étapes de calcul d'un algorithme. Ceux-ci modélisent les opérations faites par un algorithme. Dans ce langage, les sommets représentent les variables, tandis que les arrêtes représentent les opérations définies comme fonction d'une ou plusieurs variables.

Nous suivons le formalisme de *deeplearningbook* [12] qui est notre ouvrage de référence pour ce chapitre. Pour plus de détail sur les graphes de calcul ou les autres sections, nous recommandons de consulter ce manuel. De plus, les implémentations logicielles comme $tensorflow$ utilisent ces graphes pour calculer les gradients automatiquement. C'est ce qui va nous permettre de modifier l'algorithme pour apprendre les circuits quantiques.

À la figure 2.1 nous donnons deux exemples de graphes de calcul d'opération faites par un réseau neuronal. Le graphe à la gauche représente la multiplication matricielle $\mathbf{y} = \mathbf{W}^T\mathbf{x}$, alors que celui de droite représente la propagation avant pour un niveau d'un réseau neuronal $\mathbf{y} = \text{act}(\mathbf{W}^T\mathbf{x} + \mathbf{b})$.

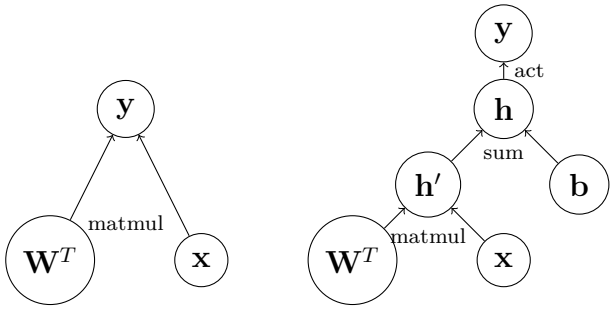


Figure 2.1. Exemples de graphes de calcul.

2.3.3. Dérivée en chaine

La règle de dérivée en chaine nous donne une recette pour calculer la dérivée d'une composition de fonctions. Comme les graphes sont des compositions d'opération, pour calculer le gradient, nous inversons la direction des ses arrêtes et appliquons cette règle pour chaque sous-graphe de sa racine aux feuilles. D'où l'idée la propagation arrière, nous propageons le gradient en sens inverse du graphe.

Définition 2.3.1 (Dérivée en chaine)**.** *Pour deux fonctions différentiables f, g, posons $z = f(g(x))$, $y = g(x)$, la règle de dérivée en chaine s'écrit :*

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

L'algorithme de rétropropagation permet de calculer les chaines de dérivées efficacement. Il applique récursivement la règle de dérivée en chaine pour chacun des sommets du graphe. Nous détaillons la méthode symbole à symbole de la rétropropagation, celle où nous ajoutons les dérivées dans le graphe de calcul, car elle est utilisée par la bibliothèque $tensorflow$ avec laquelle nous avons codé notre algorithme. Il existe une version alternative symbole à nombre utilisée par d'autres bibliothèque comme *caffe*.

Dans la figure 2.2 nous illustrons les implémentations symbole à symbole de la rétropropagation. À partir du graphe de calcul original à la gauche, celles-ci génèrent le graphe des dérivées à la droite. Ces implémentations permettent aussi de générer les dérivées de plus haut niveau. Par exemple, en appelant la procédure de génération sur le graphe de droite, nous obtenons les dérivées de second ordre.

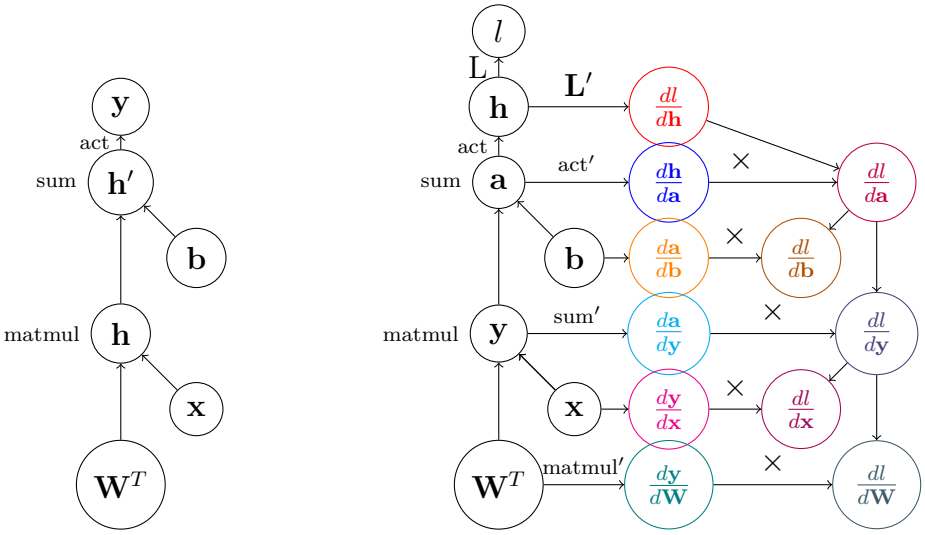


Figure 2.2. Graphe de calcul pour la rétropropagation.

La méthode symbole à symbole possède quelques avantages versus la version symbole à nombre. Tel que mentionné plus haut, elle permet de générer les directement les dérivées de plus haut niveau. Aussi, il est plus facile d'optimiser et distribuer les opérations symboliques.

Définition 2.3.2 (symbole à nombre)**.** *Pour un sous-graphe de calcul et les valeurs numériques des ses entrées, retourne les valeurs numériques de son gradient.*

Définition 2.3.3 (symbole à symbole)**.** *Pour un sous-graphe de calcul, génère un graphe étendu avec les opérations symboliques de son gradient.*

Comme nous avons des sous-graphes contenant les dérivés, nous pouvons être asynchrones avec l'ordre des opérations. Nous n'avons besoin d'évaluer le sous-graphe que lorsque sa racine est disponible. Ceci permet de dissimuler les latences de mémoire ou de distribuer le calcul. Pour éviter une surutilisation mémoire ou des calculs inutiles, les implémentations optimisent aussi pour quels sommets les dérivées sont conservées en mémoire ou recalculées. Ce type d'optimisation est plus facile avec les opérations symboliques, en conservant soit le sous-graphe ou son résultat. Elles peuvent même conserver des sous-graphes intermédiaires.

2.4. Architecture de réseaux neuronaux profonds

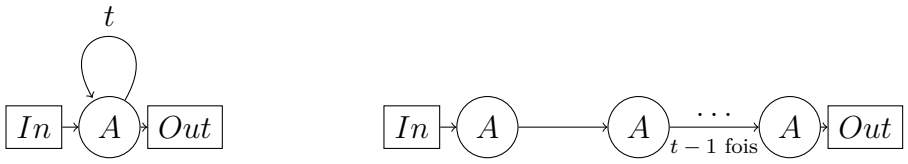
Nous passons brièvement sur différentes architectures de réseaux profonds. En premier lieu, nous expliquons les réseaux récurrents. Puis, nous montrons les réseaux à convolution.

2.4.1. Récurrence

La récurrence et la mémoire donnent une grande capacité d'apprentissage au réseaux. Celle-ci permet la modélisation de séquences et une compréhension de l'évolution de l'entrée dans le temps. D'autres architectures comme les LSTM (*Long short-term memory*) utilisent la récurrence pour obtenir de la mémoire.

Définition 2.4.1 (Réseaux neuronaux récurrents)**.** *Un réseau neuronal récurrent possède des connexions récurrentes entre ses niveaux.*

On peut voir un niveau avec t récursions, comme un réseau à propagation directe dont les t niveaux partagent les mêmes paramètres. Ils sont donc aussi vulnérables que les réseaux profonds au problème de l'explosion du gradient. Par contre, ils sont plus faciles en ressource mémoire à entrainer et à exécuter.



Les réseaux récurrents unitaires [36] sont la source de notre algorithme d'apprentissage de circuits quantique. Les auteurs utilisaient les matrices unitaires pour résoudre le problème de l'explosion et de la disparition du gradient. Comme la valeur des gradients dépend de la norme et que les matrices unitaires préservent les normes des vecteurs, cela prévient la disparation ou l'explosion du gradient.

2.4.2. Convolution

La convolution en apprentissage automatique s'inspire des filtres numériques en imagerie qui approximent la convolution mathématique formelle. De plus, en apprentissage automatique, la convolution peut aussi faire référence à la corrélation croisée qui est la convolution à un signe près. Ceci revient à un long jeu de téléphone entre la convolution telle que l'apprentissage automatique l'entend, la convolution mathématique formelle et celle que d'autres domaines définissent.

Définition 2.4.2 (Réseaux neuronaux à convolution)**.** *Réseaux neuronaux utilisant la convolution. Ils sont inspirés des neurones du cortex visuel du cerveau.*

Comme cette opération est fréquemment utilisée en imagerie, nous donnons des exemples provenant de cette tâche.

Remarque 2.4.3 (Convolution formelle)**.** *La convolution formelle est un produit de fonctions définie comme suit :*

$$c(t) = (f * w)(t) = \int f(x)w(t-x)dx$$

Pour le traitement d'image, le domaine des fonctions est en 2 dimensions ou plus. Nous appelons la fonction w un noyau que nous notons K et f correspond à l'entrée I qui est en général une image.

Exemple 2.4.4 (Convolution)**.**

$$c(t,v) = (I * K)(t,v) \sum_x \sum_y I(x,y)w(t-x,v-y)$$

par commutativité,

$$c(t,v) = (K * I)(t,v) \sum_x \sum_y I(t-x,t-y)w(x,y)$$

Exemple 2.4.5 (Corrélation croisée)**.** *En changeant le signe, ce qui revient à inverser le noyau, nous obtenons la Corrélation croisée, dénotée par le même symbole c.*

$$c(t,v) = (K * E)(t,v) \sum_x \sum_y I(t+x,t+y)w(x,y)$$

La corrélation croisée est parfois implémentée et appelée convolution dans le contexte du domaine. L'apprentissage de l'un ou de l'autre est équivalent. De haut niveau, une convolution sur image c'est balayer l'image avec un filtre comme une lampe de poche, pour obtenir des données sur l'image. Les résultats sont ensuite agrégés par du *pooling* puis passés dans une non-linéarité. Nous n'entrons pas dans le reste des détails, *pooling*, etc., car bien qu'ils ont eu beaucoup de succès, nous ne les avons pas utilisés. Comme ils sont une architecture profonde classique dans le sens traditionnel du terme, leur présentation a été faite plus par devoir que par leur nécessité à la compréhension des résultats finaux.

Ceci conclut notre présentation sur l'apprentissage automatique classique. En plus des bases, nous vous invitions fortement à bien lire la section sur la rétropogation. Celle-ci est nécessaire pour la compréhension des détails de notre l'algorithme.

Chapitre 3

Apprentissage automatique quantique

Ce chapitre explique brièvement certains algorithmes d'apprentissage automatique quantique et considère leurs faiblesses. Ces problèmes sont présents dans de nombreuses approches d'apprentissage automatique quantique. Notre approche a été motivée par le but d'obtenir des résultats concrets n'ayant pas ces faiblesses. Dans un deuxième temps, nous présentons de nouveaux algorithmes qui ont été développés pour contrer ces problèmes. Ceci permet de contraster l'état du domaine au début de ce projet comparé à celui à sa fin. En effet, nous ne fûmes pas seuls à constater les défauts présents dans les anciennes méthodes, plusieurs solutions et techniques furent développées en parallèle durant l'implémentation de ce projet.

Notre référence en début de projet et notre source d'introduction dans le domaine fut la revue de littérature *Quantum Machine Learning* [8], en version originale de 2016. Les auteurs la modernisèrent en 2018, dans laquelle ils discutent des nouveaux développements sur les méthodes déjà présentées. Tiré de leur résumé de 2018 : « *Recent work has made clear that the hardware and software challenges are still considerable but has also opened paths towards solutions* ».

C'est dans cet article que nous avons trouvé initialement les algorithmes que nous présentons en première partie. Sans plus tarder, nous entrons de plain-pied dans HHL, algorithme que nous ne pouvions pas laisser passer, car il a une grande réputation que ce soit en bon ou en mauvais dans le domaine.

3.1. HHL

HHL tient son nom des initiales de ses auteurs Aram W. Harrow, Avinatan Hassidim et Seth Lloyd qui le développèrent pour résoudre des systèmes d'équations linéaires dans leur papier de 2008 : *Quantum algorithm for solving linear systems of equations* [14]. Il fut rapidement étendu à l'analyse en composant principal [18] et aux machines à vecteur de support [23].

L'algorithme HHL résout un système d'équations linéaires. C'est-à-dire pour A, une matrice $n \times n$, et b un vecteur, HHL approxime un x tel que

$$Ax = b$$

en temps $O(\log(n))$.

3.1.1. Les petits caractères

Nous avons commencé à regarder HHL et ses algorithmes dérivés dû à leur promesse de gain exponentiel. Par contre, avec lecture approfondie nous trouvions à cet algorithme plusieurs clauses échappatoires. C'est-à-dire que HHL nécessite beaucoup de conditions pour obtenir le gain exponentiel promis. Nous n'avons présentement aucune idée si ces conditions peuvent survenir en pratique. Par la suite, nous avons été mis au courant à propos de l'article de Scott Aronson *Quantum Machine Learning Algorithms : Read the Fine Print*[1]. Cet article présente les défauts que nous avions trouvés et d'autres.

(1) Mémoire quantique. L'algorithme nécessite une mémoire quantique rendant accessible une superposition des adresses en temps logarithmique, pour préparer $|b\rangle$ et ne pas perdre son avantage exponentiel. Il est possible pour certains types de b spécifique, par exemple si l'on peut les calculer, d'éviter la mémoire quantique et de les préparer à l'aide d'un circuit. D'autres types de b par contre restent exponentiels à encoder, car ils contrediraient sinon l'optimalité de l'algorithme de Grover [7].

(2) Matrice creuse. Pour effectuer l'exponentiation rapide e^{-iAt} de la matrice A, cette dernière doit être une matrice creuse.

(3) Conditionnement de la matrice. A doit non seulement être inversible, mais avoir un bon conditionnement. HHL dépend de $K = \frac{|\lambda_{max}|}{|\lambda_{min}|}$.

(4) Borne de Holevo. La sortie $|x\rangle$ est un encodage approximatif de x sur les amplitudes de $\log n$ qubits. Nous pouvons extraire stochastiquement au plus $\log n$ bits d'information de celui-ci en une itération. Trouver la valeur de l'amplitude spécifique correspondant à une des composantes x_i du vecteur x demanderait n répétitions, ce qui éliminerait le gain exponentiel.

Nous avions identifié les deux premières conditions. Nous les trouvions suffisantes pour rejeter l'idée d'évaluer HHL empiriquement. L'algorithme est non seulement fortement spécialisé, mais aussi, il est possible que les cas auxquels il s'applique aient un algorithme classique aussi rapide sans que ce soit le résultat du siècle en complexité du calcul. Conséquemment, il faut des processeurs quantiques à grande échelle pour évaluer HHL. Certainement, si la mémoire quantique devient possible HHL devient alors plus réaliste. Notons que la quatrième condition, la borne de Holevo, affecte tous les algorithmes quantiques.

3.2. Quantization d'architecture classique

Par rapport à la quantization de beaucoup d'algorithmes d'apprentissage machine classiques, nous sommes très sceptiques. Dans une première catégorie, nous retrouvons les algorithmes utilisant HHL en sous-routine tels les *SVM* [23]. Leurs évaluations se rapportent à celle de HHL. Dans une seconde catégorie, nous avons les algorithmes utilisant la descente de gradient quantique. Pour ces méthodes, il est difficile d'évaluer leur efficacité et leur accélération pratique sans matériel quantique. Parmi celles-ci nous étions familiers avec la généralisation quantique de réseaux à propagation avant [32]. Les auteurs donnent une efficacité polynomiale à leur descente de gradient, mais en pratique plusieurs méthodes polynomiales sont trop inefficaces pour être utilisées, comme les méthodes de Newton. Notons que les algorithmes de descente de gradient quantique peuvent encore être améliorés et que ceux-ci s'appliquent à l'apprentissage de circuit quantique. Vu l'importance de la descente de gradient quantique, nous lui dédions une section à elle seule plus loin dans ce chapitre.

3.2.1. Machine de Boltzmann

Dans une troisième catégorie, nous trouvons les machines de Boltzmann quantiques [3]. Contrairement aux autres quantizations, pour les machines de Boltzmann ce fut un retour

aux sources. Inspirée originellement de la physique, elle ressemble à la physique quantique derrière les circuits. Nous les expliquerons en peu de détail dû à leur complexité. Les manuels d'apprentissage automatique couvrent très peu les versions classiques, leur difficulté d'entrainement causant une faible utilisation en pratique. Il s'agit vraisemblablement du seul algorithme aussi difficile à entrainer classiquement que quantiquement, ce qui est à l'avantage du quantique. L'article *Generative training of quantum Boltzmann machines with hidden units* [34] de Nathan Weibe donne l'entrainement des versions quantiques. Pour les versions classiques, le chapitre 20 du *deeplearingbook* [12] les couvre.

À notre humble avis, nous pensons qu'elles sont vouées à un avenir beaucoup plus prometteur en quantique qu'en classique.

3.3. Circuit quantique variationnel

Nous allons maintenant dans la deuxième phase de ce chapitre, où nous présentons des résultats plus récents. Les circuits quantiques variationnels apprennent un circuit quantique avec une mise à jour quantique. C'est-à-dire que la descente de gradient utilise le quantique dans une sous-routine pour faciliter l'apprentissage du circuit. Ils sont introduits une première fois dans *Circuit-centric quantum classifiers* [25], où ils apprennent une tâche de classification binaire. Pour la résoudre, les auteurs utilisent un encodage en amplitude pour envoyer des données classiques de grande dimension au circuit quantique. Ensuite, ils donnent une procédure de descente de gradient hybride pour mettre à jour le circuit.

Les différentes topologies de circuit consistant de rotations et de portes contrôlées sont plus terre à terre avec les ordinateurs physiques, ce qui a permis d'obtenir des résultats utilisant les processeurs quantiques [13]. Différents modèles basés sur les circuits quantiques sont présentés dans cette revue de littérature *Parameterized quantum circuits as machine learning models* [5], dans lequel les résultats d'une dizaine d'expériences sur les processeurs quantiques ont été compilés.

3.4. Descente de gradient quantique

La descente de gradient quantique consiste à utiliser un calcul quantique pour obtenir les dérivées. Nous incluons les méthodes hybrides. Elle peut aussi s'appliquer aux modèles

classique ou hybride. Nous présentons les techniques une et deux de descente de gradient quantique de *Evaluating analytic gradients on quantum hardware* [24]. Quoiqu'il y en existe d'autres, elles s'appliquent à l'apprentissage de circuits quantiques. Nous ne passons pas sur les détails et les preuves de ces techniques.

3.4.1. Décalage de paramètre

Le décalage de paramètre est limité au cas où les matrices génératrices ont deux valeurs propres uniques $\pm r$. C'est-à-dire que pour une matrice unitaire U paramétrée par θ_i tel que $U(\theta_i) = e^{-i\theta_i H}$, nous avons H la matrice Hermitienne appelée son générateur à deux valeurs propres uniques $\pm r$. Posons $f(x;\theta) = \langle x|C_\theta^\dagger O C_\theta|x\rangle$ comme l'espérance des probabilités de mesure avec l'observable O des qubits à la sortie du circuit quantique paramétré par θ avec l'entrée x. Alors, pour le paramètre d'une porte θ_i,

$$\frac{\partial f(x;\theta_i)}{\partial \theta_i} = (f(x;\theta_i + s) - f(x;\theta_i - s))r,$$

avec $s = \frac{\pi}{4r}$. Pour estimer la dérivée de f par rapport au paramètre d'une porte, nous n'avons qu'a générer deux nouveaux circuits, un où la porte est décalée par $+s$ et un autre où la même porte est décalé par $-s$, puis estimer l'espérance de leur sortie. À partir du circuit original, cette tâche est assez simple, mais elle doit être faite pour tous les paramètres θ du circuit. Cette règle s'applique aussi sur plusieurs autres cas spéciaux comme les rotations sur un qubit avec des décalages s spécifiques à ces cas.

3.4.2. Combinaison linéaire de portes unitaires

Cette deuxième méthode s'occupe des cas ou le décalage ne s'applique pas. Dans un premier temps nous reformulons $f(x;\theta)$ pour isoler la porte G_{θ_i} que nous entrainons. Pour ce faire, nous pouvons déplacer les exécutions avant celle-ci dans l'état de l'entrée $|x\rangle \to |\phi\rangle$ et déplacer les exécutions la suivant dans l'observable $O \to P$.

$$f(x;\theta_i) = \langle\phi|G_{\theta_i}^\dagger P G_{\theta_i}|\phi\rangle$$

La dérivée s'obtient par la règle du produit :

$$\frac{\partial f(x;\theta_i)}{\partial \theta_i} = \frac{\partial \langle\phi|G_{\theta_i}^\dagger P G_{\theta_i}|\phi\rangle}{\partial \theta_i} = \langle\phi|\frac{\partial G_{\theta_i}^\dagger}{\partial \theta_i} P G_{\theta_i}|\phi\rangle + \langle\phi|G_{\theta_i}^\dagger P \frac{\partial G_{\theta_i}}{\partial \theta_i}|\phi\rangle$$

Avec quelques manipulations d'algèbre linéaire nous avons :

$$\langle\phi|\frac{\partial G_{\theta_i}^\dagger}{\partial\theta_i}PG_{\theta_i}|\phi\rangle + \langle\phi|G_{\theta_i}^\dagger P\frac{\partial G_{\theta_i}}{\partial\theta_i}|\phi\rangle = \frac{1}{2}\langle\phi|\left(\frac{\partial G_{\theta_i}}{\partial\theta_i} + G_{\theta_i}\right)^\dagger P\left(\frac{\partial G_{\theta_i}}{\partial\theta_i} + G_{\theta_i}\right)|\phi\rangle - \frac{1}{2}\langle\phi|\left(\frac{\partial G_{\theta_i}}{\partial\theta_i} - G_{\theta_i}\right)^\dagger P\left(\frac{\partial G_{\theta_i}}{\partial\theta_i} - G_{\theta_i}\right)|\phi\rangle$$

Comme $\frac{\partial G_{\theta_i}}{\partial\theta_i}$ n'est pas nécessairement unitaire on ne peut pas l'évaluer directement sur un processeur quantique. Puisqu'elle est une matrice complexe carrée, on peut la décomposer en combinaison linéaire de matrices unitaires à coefficients réels, lesquelles sont exécutables sur un processeur quantique.

$$\frac{\partial G_{\theta_i}}{\partial\theta_i} = \sum_k \alpha_k A_k$$

Cette combinaison linéaire peut se limiter à deux matrices unitaires i.e $k = 2$ de cette façons :

$$\frac{\partial G_{\theta_i}}{\partial\theta_i} = \frac{\alpha}{2}((A_0 + A_0^\dagger) + i(A_1 + A_1^\dagger))$$

la dérivée devient donc,

$$\begin{aligned}\frac{\partial f(x;\theta_i)}{\partial\theta_i} &= \sum_k \alpha_k(\langle\phi|A_k PG_{\theta_i}|\phi\rangle + \langle\phi|G_{\theta_i}^\dagger PA_k|\phi\rangle) \\ &= \sum_k \alpha_k\Big(\left(\langle\phi|A_k + G_{\theta_i}\right)^\dagger P\left(A_k + G_{\theta_i}\right)|\phi\rangle - \frac{1}{2}\langle\phi|\left(A_k - G_{\theta_i}\right)^\dagger P\left(A_k - G_{\theta_i}\right)|\phi\rangle\Big)\end{aligned}$$

Nous devons simuler chaque A_k puis effectuer leur combinaison. Lorsque $k = 2$, une généralisation du test de Hadamard permet de calculer la combinaison. Ceci est l'idée générale. Nous ne présentons pas tous les détails, les curieux peuvent allez plus loin en lisant notre source [24].

Notre référence, *Evaluating analytic gradients on quantum hardware* [24] présente d'autres techniques adaptées aux systèmes quantiques à variable continue. Ces systèmes relèvent plus de la physique, si bien que nous ne les couvrirons pas. *Universal Training Algorithm for Quantum Deep Learning* [31] introduit d'autres techniques pour l'entrainement de circuits quantiques adaptables aussi à l'entrainement de modèles classiques. Ces algorithmes sont

de plus haut niveau et utilisent l'exponentiation rapide de matrice présente dans HHL. Ils souffrent donc des conditions 2 et 3 de l'algorithme de HHL.

Cette brève revue de littérature a pour objectif de donner le contexte entre l'état de l'art au début de ce projet et celui à la fin. Elle couvre les résultats importants en commençant par HHL, puis les circuits variationnels et la descente de gradient quantique qui sont arrivés durant l'implémentation de ce mémoire. Principalement, comme nous codions une méthode d'apprentissage de circuit quantique, nous fûmes très contents d'apprendre l'existence des circuits variationnels. Bien que notre méthode risque fortement de devenir obsolète suite à l'apparition d'ordinateurs quantiques à grande échelle, le modèle qu'elle permet d'apprendre, les circuits quantiques, pourra continuer à être utilisé à l'aide des algorithmes d'apprentissage développés pour des circuits variationnels.

Chapitre 4

Apprentissage de circuit quantique

4.1. Introduction

Les circuits quantiques effectuent l'opération $|\phi\rangle = U|\psi\rangle$, où U est la matrice unitaire paramétrant le circuit, $|\psi\rangle$ son entrée et $|\phi\rangle$ sa sortie. Mathématiquement, cette opération est équivalente à celle d'un *SVM* ou un réseau de neurones linéaire $\mathbf{y} = A\mathbf{x}$. Notre objectif sera de trouver cette matrice, pour une tâche donnée. Nous étudions la tâche de classification, dans le but de montrer le fonctionnement de l'algorithme. Pour apprendre le circuit, nous nous inspirons des réseaux de neurones classiques en utilisant la descente de gradient.

4.2. Algorithme d'apprentissage

Pour pouvoir conserver un circuit quantique lors de notre apprentissage, les matrices doivent rester unitaires. La variété de Stiefel correspond à l'ensemble de ces matrices. Un algorithme permettant la descente de gradient sur cet espace fut utilisé par Scott Wisdom et autres [36] pour apprendre des réseaux de neurones récurrents. Leur implémentation se base sur les notes de Tagare [30] sur ce sujet.

Définition 4.2.1 (Variété de Stiefel)**.** *Nous définissons la variété de Stiefel comme l'ensemble continu des matrices unitaires.*

$$V_n(\mathbb{C}^n) = \{U \in \mathbb{C}^{n\times n} : UU^\dagger = U^\dagger U = \mathbf{I}\}$$

Notre algorithme d'apprentissage se base sur l'optimisation sur la Variété de Stiefel. Le premier algorithme (algorithme 5) utilise les modifications faites par Scott Wisdom sur

l'algorithme de Tagare [30]. Le deuxième algorithme (algorithme 6) adapte les travaux d'optimisation de Jonathan Siegel [28] et utilise l'algorithme 5 en sous-routine.

Algorithme 5 : Optimisation sur la variété de Stiefel (Algorithme de Tagare)

L'algorithme reçoit en entrée la matrice unitaire $\mathbf{W}$

Calcul du gradient de la fonction de perte L en fonction de la matrice $\mathbf{W}$.

$\mathbf{G} = \frac{\partial L}{\partial \mathbf{W}}$

Calcul de la tangente.

$\mathbf{A} = \mathbf{G}^\dagger \mathbf{W} - \mathbf{W}^\dagger \mathbf{G}$

Calcul de la transformée de Cayley multipliée par la moitié du taux d'apprentissage.

$\mathbf{C} = \left(\mathbf{I} - \frac{\lambda}{2}\mathbf{A}^{(\mathbf{k})}\right)^{-1}\left(\mathbf{I} + \frac{\lambda}{2}\mathbf{A}^{(\mathbf{k})}\right)$

Calcul de la transformée de Cayley multipliée par la matrice originale.

$\mathbf{Y}^{(k)}(\lambda) = \mathbf{C}\mathbf{W}^{(k)}$

Mise à jour de la matrice.

$\mathbf{W}^{(k+1)} = \mathbf{Y}^{(k)}(\lambda)$

Pour l'algorithme 6, nous avons besoin du 5 en sous-routine. La lecture de la thèse de Jonathan Siegel [28] nous a appris que l'algorithme 5 effectue mathématiquement une rétraction et qu'il est donc plus générique que nous le pensions. Nous définissons alors comment l'appeler en sous-routine de cette façon : $tagare(\mathbf{W} = Var_0, \mathbf{G} = Var_1, \lambda = Var_2)$

4.2.1. Momentum de Nesterov

Pour accélérer la convergence, nous utilisons une variante du momentum de Nesterov découverte par Jonathan Siegel [28] dans sa thèse de doctorat. L'idée principale de son algorithme est de pouvoir effectuer une moyenne pondérée de deux points sur la variété, donc de calculer $(1-\alpha)X + \alpha Y$.

Algorithme 6 : Momentum de Nesterov sur la variété de Stiefel

Pour le momentum de Nesterov, on met à jour les paramètres au début

$\mathbf{X}_{t+1} = tagare(\mathbf{W} = \mathbf{X}_t, \mathbf{G} = \nabla \mathbf{X}_{t+1}, \lambda = \lambda)$

Calcul du momentum

$\mathbf{V}_t = 2\mathbf{X}(\mathbf{I} + \mathbf{X}_{t+1}^\dagger \mathbf{X}_t)$

Projection sur l'espace tangent dual (optionnel)

$\mathbf{V}_t = \mathbf{V}_t - \frac{1}{2}\mathbf{X}^\dagger(\mathbf{V}^\dagger \mathbf{X_t} - \mathbf{X_t}^\dagger \mathbf{V})$

Application du momentum

$Y_{t+1} = tagare(\mathbf{W} = \mathbf{V}_t, \mathbf{G} = \mathbf{Y}_t, \lambda = \mu)$

Tout comme Scott Wisdom et autres [36] ont fait avec la méthode de Tagare [30] pour l'algorithme 5, notre variante ignore les conditions de redémarrage et de Armijo-Wolfe. L'étape de projection sur l'espace tangent dual fut récemment ajoutée pour corriger des erreurs numériques lorsque l'algorithme est proche du minimum. Toutefois, notre implémentation ne l'utilise pas, car elle n'existait pas lorsque nous l'avons codée. Par contre, elle pourrait être utile. Il y a quelques cas où nous pensons qu'elle aiderait, principalement dans l'apprentissage d'algorithmes, surtout ceux déjà connus et dans certains cas où nous sommes très proches d'un minimum.

4.2.2. Fonction de coût et encodage de la sortie

Pour pouvoir choisir un objectif, nous devons interpréter les résultats du circuit quantique. Les encodages ci-dessous s'appliquent principalement à la classification binaire et multi-classes. Certains d'entre eux s'appliquent aussi dans l'approximation d'un opérateur quantique, ou dans l'apprentissage de protocole quantique comme la téléportation.

Le paradigme quantique ajoute une nouvelle contrainte au choix de la fonction de perte, à savoir la préservation des propriétés quantiques du circuit. L'erreur quadrique moyenne, une fonction de perte traditionnelle en apprentissage machine classique, ignore les états quantiques indiscernables. En effet, un facteur de phase globale augmente la perte sans que le résultat final change.

Définition 4.2.2 (Erreur quadratique moyenne)**.** *Pour deux vecteurs complexes t et o,*

$$MSQ(t,o) = \sum_{i=0}^{n-1} |t_i - o_i|^2$$

Exemple 4.2.3. *Les états* $|\psi\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ *et* $|\phi\rangle = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$ *sont indiscernables. Par contre,* $MSQ(|\psi\rangle,|\phi\rangle) = 4$*. Une bonne fonction de perte aurait un coût de zéro pour deux états indiscernables.*

Pour préserver ces propriétés, nous suggérons une approche basée sur la fidélité.

Définition 4.2.4 (Fidélité)**.** *Pour deux états ρ, σ*

$$F(\rho,\sigma) = \mathrm{tr}\left(\sqrt{\rho^{\frac{1}{2}}\sigma\rho^{\frac{1}{2}}}\right)$$

Remarque 4.2.5. *Pour deux états purs* $|\psi\rangle$, $|\phi\rangle$

$$F(|\psi\rangle,|\phi\rangle) = |\langle\psi|\phi\rangle|^2$$

Cette fonction préserve les propriétés quantiques du circuit. Par contre, dans le cas où $|\psi\rangle$ et $|\phi\rangle$ n'ont pas le même nombre de qubits, elle n'est pas définie. Nous créons donc une nouvelle fonction basée sur celle-ci à l'aide du produit scalaire partiel.

Définition 4.2.6 (Produit scalaire partiel)**.** *Pour* $|\psi\rangle$, $|\phi\rangle$ *sur registres AB et B.*

$$\langle\phi_B|\psi_{AB}\rangle = (\mathbb{I}_A \otimes \langle\phi|_B)|\psi\rangle_{AB}$$

Définition 4.2.7 (Fidélité partielle)**.** *Pour* $|\psi\rangle$, $|\phi\rangle$ *sur registres AB et B où n est la dimension de A.*

$$FP(|\psi\rangle,|\phi\rangle) = \sum_{i=0}^{n-1} |\langle\phi_B|\psi_{AB}\rangle_i|^2$$

La fidélité partielle extrait, dans le cas où les états sont purs, la probabilité de mesurer $|\phi\rangle$ sur le registre B. En maximisant cette probabilité, nous pouvons trouver un bon circuit.

Remarque 4.2.8. *Puisque* $\mathbb{I}_A$ *est une matrice identité, nous pouvons exprimer la fidélité partielle de cette façon équivalente : pour* 1_n *un vecteur* $1 \times n$ *de 1,*

$$FP(|\psi\rangle,|\phi\rangle) = |(1_n \otimes \langle\phi|_B)|\psi\rangle_{AB}|^2 .$$

Cette reformulation permet un calcul plus efficace en mémoire.

Définition 4.2.9 (*Softmax*)**.**

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=0}^{n-1} e^{z_j}}$$

Nous pouvons interpréter l'état mixte sur l'espace des qubits correspondant aux classes comme étant la probabilité d'être membre de chaque classe. Nous obtenons un encodage équivalent au *softmax* classique. Cela nous permet de généraliser la fonction de perte utilisée traditionnellement dans les réseaux neuronaux, à savoir l'entropie croisée.

Définition 4.2.10 (Entropie croisée)**.**

$$L(y,p) = \sum_{i=0}^{n-1} -y_i \log(p_i)$$

Définition 4.2.11 (*one-hot*)**.** *L'encodage one-hot pour la classification consiste à présenter chaque classe à la sortie par un vecteur consistant de zéros et d'un seul 1 à la position de la*

valeur de la classe. Par exemple, la classe 15 dans une classification à 100 classes aurait un vecteur ayant un 1 à la position 15 et zéro aux 99 autres positions.

Notre interprétation équivaut au *softmax* avec encodage *one-hot* ayant un *padding* de classe additionnel, car le nombre maximum d'amplitudes possibles sur un état est $2^{nbqubit}$. L'entropie croisée est équivalente à la log probabilité dans les expériences exécutées.

4.2.3. Descente de gradient

L'apprentissage par descente de gradient se fait par rétropropagation. Cela nous permet d'utiliser les bibliothèques d'apprentissage automatique classique supportant les gradients complexes, comme *tensorflow*.

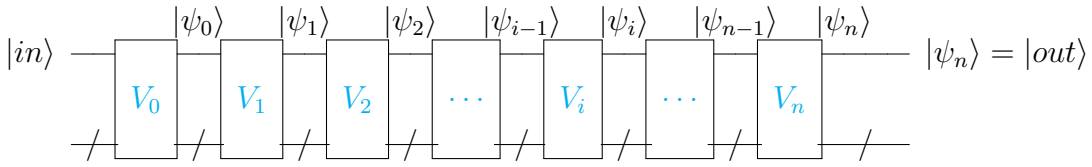


Figure 4.1. Circuit quantique.

Considérons le circuit de la figure 4.1. Avec L, la fonction de perte choisie, nous obtenons par les règles de dérivée en chaines les gradients de la table 4.1

Tableau 4.1. Table des gradients.

$$\frac{\partial L}{\partial V_n} = \frac{\partial L}{\partial \psi_n}\frac{\partial \psi_n}{\partial V_n} \quad \frac{\partial L}{\partial \psi_i} = \frac{\partial L}{\partial \psi_{i+1}}\frac{\partial \psi_{i+1}}{\partial \psi_i}$$
$$\frac{\partial L}{\partial V_i} = \frac{\partial L}{\partial \psi_i}\frac{\partial \psi_i}{\partial V_i} \quad \frac{\partial L}{\partial \psi_{i-1}} = \frac{\partial L}{\partial \psi_i}\frac{\partial \psi_i}{\partial \psi_{i-1}}$$

Le gradient se calcule par rétropropagation comme un réseau neuronal classique avec des matrices complexes et sans non-linéarité. Une fois les gradients calculés, l'algorithme 5 met à jour les matrices unitaires. Comme *tensorflow* utilise les graphes de calcul il n'y aucune difficulté additionnelle à obtenir les gradients. Dans la version en langage C++ utilisant la librairie d'algèbre linéaire *eigen*, ceux-ci ont été codés manuellement.

4.3. Porte quantique

Nous définissons une porte quantique de notre circuit par sa matrice unitaire correspondante. Leur simulation se fait en espace exponentiel sur le nombre de qubits $O(4^n)$. Comparé à l'espace $O(2^n)$ utilisé par les états, ceci devient en pratique le premier facteur limitant de la simulation.

4.3.1. Optimisation de mémoire

Le théorème de Solovay–Kitaev, théorème 1.3.10 du chapitre 1, permet d'approximer une porte quantique sur un grand nombre de qubits par plusieurs portes sur un petit nombre de qubits. La paramétrisation du circuit par de petites matrices unitaires nécessite d'effectuer l'opération suivante efficacement

$$|out\rangle = (\mathbb{I}_f \otimes U_q \otimes \mathbb{I}_p)|input\rangle$$

Cette opération correspond à une porte de q qubits appliquée sur un état de $f+q+p=n$ qubits tel qu'indiqué dans le diagramme ci-dessous.

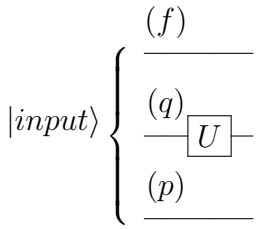


Figure 4.2. Circuit montrant l'application d'une porte quantique

Calculer $\mathbb{I}_f \otimes U_q \otimes \mathbb{I}_p$ avant de faire la multiplication matrice vecteur revient au cas où on doit avoir en mémoire une matrice de 4^n nombres complexes. Bien que la matrice soit stockée temporairement, nous obtenons plus d'efficacité en calculant la multiplication directement par l'algorithme suivant.

Algorithme 7 : Calcul efficace en mémoire de l'application d'une petite porte sur un grand état en évitant le calcul du produit tensoriel.

entrée : $|input\rangle$, un état quantique de n qubits

sortie : $|out\rangle$, l'application de la porte U sur cet état

$F = 2^f$, $Q = 2^q$, $P = 2^p$

$|out\rangle$: initialisé a zéro

for $k = 0$ **to** $F - 1$ **do**

  $x = k \times Q \times P$

  for $col = 0$ **to** $Q - 1$ **do**

    $y = x \times P \times col$

    for $j = 0$ **to** $Q - 1$ **do**

      for $i = 0$ **to** $P - 1$ **do**

        $|out\rangle[x + i + j \times P] = U_{j,col} \times |input\rangle[y + i]$

return $|out\rangle$

La dérivée est calculable facilement. Les deux algorithmes se ressemblent beaucoup car il s'agit de la même boucle. La différence entre les deux est la ligne dans le cœur de la boucle qui dans l'algorithme suivant est la dérivée du précédent.

Algorithme 8 : Calcul efficace en mémoire de la dérivée de l'application d'une petite porte sur un grand état.

entrée : *prev*, gradient du niveau précédent

sortie : *GradU*, gradient de le porte U

$F = 2^f$, $Q = 2^q$, $P = 2^p$

GradU : initialisé a zéro

for $k = 0$ **to** $F - 1$ **do**

  $x = k \times Q \times P$

  for $col = 0$ **to** $Q - 1$ **do**

    $y = x \times P \times col$

    for $j = 0$ **to** $Q - 1$ **do**

      for $i = 0$ **to** $P - 1$ **do**

        $GradU_{j,col} = \langle input|[y + i] \times prev[x + i + j \times P]$

return *GradQ*

Pour l'implémentation *tensorflow* de l'algorithme, nous utilisons leur sous-routine *einsum* basée sur la notation de Einstein. Celle-ci généralise ce type de boucle imbriqué sur des variables SIMD. Le gradient automatique de *tensorflow* s'occupe même de calculer automatiquement la dérivée pour nous.

Pour visualiser l'algorithme 7, il s'agit d'une association entre les valeurs de la matrice U_q et où elles seraient présentes dans le produit tensoriel $\mathbb{I}_f \otimes U_q \otimes \mathbb{I}_p$ calculé explicitement. La deuxième boucle sert à associer les valeurs de la matrices U_q à leur emplacement dans le produit tensoriel $U_q \otimes \mathbb{I}_p$. La première boucle sert à associer les valeurs de ce produit tensoriel $(U_q \otimes \mathbb{I}_p)$ à leur emplacement dans le premier produit tensoriel dans $\mathbb{I}_f \otimes (U_q \otimes \mathbb{I}_p)$. Les deux dernières boucles consistent à faire la multiplication matrice vecteur avec les bonnes valeurs du vecteur en entrée.

4.3.2. Hyperparamètre

La nécessité pratique d'utiliser un ensemble de petites portes ajoute un hyperparamètre majeur : la topologie du circuit. La capacité du circuit à apprendre dépend fortement de sa forme. Nous avons créé deux types de topologie pour l'entrainement de circuit. La topologie remplie puis la topologie en quinconce.

Définition 4.3.1 (Topologie remplie)**.** *Dans une topologie remplie, nous maximisons la taille des portes. Le premier niveau est rempli de portes de largeur maximale, avec une porte de plus petite arité si nécessaire pour les qubits restants. Puis le second niveau commence par une porte d'un qubit, suivie du remplissage par des portes de largeur maximale. En général on pourrait utiliser ici une de première porte de plus d'un qubit. Ces niveaux alternent un à la suite de l'autre.*

Définition 4.3.2 (Topologie en quinconce)**.** *Inspiré de la formation des légionnaires romains : la quinconce. Cette topologie a un premier niveau rempli, puis un second niveau décalé de la moitié de l'arité maximale avant d'être rempli de porte d'arité maximale. Toutefois, pour des raisons d'optimisation du temps de calcul, nous n'insérons pas de porte autant pour les premiers qubits du décalage que pour traiter les qubits qui pourrait rester après la dernière porte d'arité maximale sur le second niveau. Ces niveaux alternent encore une fois un à la suite de l'autre.*

4.3.3. Compromis

En pratique, le calcul matriciel est fortement parallélisé, ce qui cause des compromis en temps de calcul entre les portes très petites et celles ayant un nombre moyen de qubits. De plus, l'un des objectifs dans l'apprentissage du circuit est de pouvoir l'exécuter, une fois

entrainé, sur un véritable ordinateur quantique physique. Présentement, ceux-ci préfèrent des portes de deux qubits ou moins. Lors de la simulation quantique, un circuit paramétré avec ces portes perd beaucoup de parallélisation. Pour conserver la parallélisation à la simulation, nous utilisons des portes de taille de quatre à dix qubits, huit qubits étant la taille la plus fréquente. Nous voyons deux options principales permettant l'exécution de ce circuit sur un ordinateur quantique. Premièrement, appliquer l'algorithme de Solovay–Kitaev pour approximer les portes du circuit. Deuxièmement, utiliser notre algorithme pour approximer chaque porte individuellement. La meilleure des approches sera décidée si l'occasion se présente, le travail dépassant les limites de ce projet.

4.3.4. Encodage de l'entrée

Nous utilisons l'encodage de *Schuld* et collaborateurs [25] pour l'entrée du circuit. Nous transformons les entrées en vecteurs normalisés pour les encoder dans les amplitudes d'un qubit. Lorsque possible, nous répétons la même entrée plusieurs fois pour profiter des non-linéarités tel que suggéré par le même article.

4.3.5. Métriques

Nous utilisons deux métriques pour évaluer la performance de notre modèle : précision majoritaire et précision maximale.

Définition 4.3.3 (Précision majoritaire)**.** *La précision majoritaire est la proportion des données sur lesquelles la probabilité que la bonne classe soit mesurée est supérieure à 50%.*

Définition 4.3.4 (Précision maximale)**.** *La précision maximale est la proportion des données sur lesquelles la probabilité que la bonne classe soit mesurée est supérieure à celle des autres classes.*

La précision maximale est la même métrique que la précision utilisée en apprentissage classique. Cependant, elle ignore l'aspect stochastique de la mesure quantique, car elle indique seulement que si on échantillonnait le circuit à répétition, il se dirigerait vers la bonne réponse. La précision majoritaire est équivalente à la précision maximale dans le cas binaire. Elle montre les cas pour lesquels le circuit en général nécessite peu d'échantillonnage pour obtenir la bonne réponse.

4.4. Modèles

L'algorithme est utilisé pour apprendre des circuits quantiques pour deux modèles d'apprentissage machine. Soit un circuit quantique pur, soit un modèle hybride où le circuit quantique reçoit ses entrées d'un modèle classique, comme un réseau neuronal, pour lequel tout autre modèle classique utilisant la descente de gradient devrait être compatible.

4.4.1. Circuit quantique

Le modèle reçoit son entrée encodée sur les amplitudes d'un état. L'algorithme entraine le circuit pour résoudre une tâche, principalement de la classification dans notre cas.

4.4.2. Approche hybride

Le modèle reçoit son entrée sans nécessiter d'encodage particulier. La partie classique produit une description de l'état quantique par ses amplitudes, cet état est donné en entrée au circuit quantique. La descente de gradient permet d'entrainer les deux parties en coopération. La figure 4.3 donne le schéma du modèle.

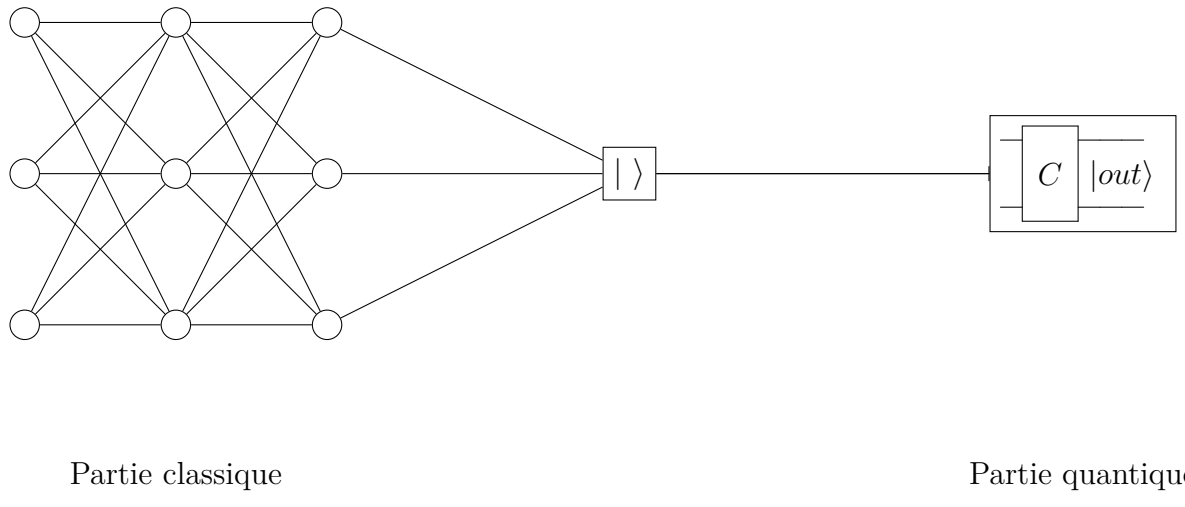


Figure 4.3. Modèle hybride.

Propriétés	pur	linéaire (perceptron)	non-linéaire (réseaux de neurone)	Autoencodeur
Capacité classique	non	minime	forte à très forte	ajustable
Réduction de dimensionnalité	non	oui	oui	oui

Figure 4.4. Propriétés du modèle selon la partie classique.

Pour juger de l'approche hybride, il est nécessaire de connaitre la capacité d'apprentissage de la partie classique. En effet, il est nécessaire de caractériser en quoi le circuit quantique aide l'apprentissage. Par exemple, pour *mnist* il est très facile pour un réseau simple à deux couches d'obtenir une précision de 98,5%. Conséquemment, le circuit quantique doit justifier sa présence en obtenant de meilleures performances que la partie classique seule.

Dans la figure 4.4, nous illustrons la capacité de différentes approches classiques. Clairement, les ensembles de données ayant des exemples de grande taille nécessitent une réduction de dimensionnalité, pour pouvoir être simulés. Dans ces cas, l'approche hybride doit être utilisée. Par contre, le choix de la partie classique doit être fait judicieusement. À cause de la croissance exponentielle en mémoire, certains ensembles de données triviaux à apprendre pour des modèles classiques, comme *mnist* mentionné précédemment, deviennent non triviaux pour l'apprentissage de circuit quantique.

4.5. Expériences

Pour tester le bon fonctionnement du code nous utilisons les données iris. Celles-ci sont constituées des mesures de longueur et largeur des pétales et sépales de spécimens de la fleur iris pouvant appartenir à trois espèces d'iris différentes : *Iris setosa, Iris versicolor* et *Iris virginica*. La tâche consiste à classifier les exemples dans l'une de ces trois classes. La faible taille des données permet d'arriver à de très bons résultats sans nécessiter une très grande puissance de calcul. Par contre, le faible nombre de données dans l'ensemble d'entrainement (150) et de test (30) rend une conclusion sur les performances du modèle difficile. Il est facile d'avoir les 30 exemples de test à 100% avec un peu de chance, ce qui est le cas pour les deux métriques de notre modèle.

Le code de l'implémentation *tensorflow* de l'algorithme et des expériences se trouve sur *Github* au lien suivant : `https://github.com/AldoLamarre/quantumcircuitlearning`.

4.5.1. Matériel

Nous disposons à la fois d'un CPU et d'un GPU. Le CPU utilisé est un threadripper 1950x de 16 cœurs physiques et 32 cœurs logiques avec 64GB de RAM en mode NUMA (*non-uniform memory access*). Il est possible qu'en mode UMA (*uniform memory access*) le CPU obtienne de meilleures performances, mais il sera plus lent que le GPU sauf pour de

petits circuits. Nous pensons que NUMA est préférable pour réduire la latence entre le GPU et le *Zeppelin die* contenant le PCI-E I/O communiquant avec le GPU. Le GPU est un Titan Xp avec 12GB de RAM. Il nous faudrait plus de RAM sur le GPU, mais les configurations à plus grandes capacités devenaient rapidement en dehors de notre budget.

Malheureusement, nous n'avons pas optimisé le transfert mémoire entre le CPU et le GPU. C'est pour cela que le CPU peut apprendre certains circuits sur lesquels le GPU *crash* par manque de mémoire. Comme les circuits quantiques sont extrêmement longs à entrainer et que la différence entre 12GB et 64GB sur une fonction exponentielle est minime, nous n'avons pas cherché à trouver ce type de circuit limite où l'entrainement par le CPU fonctionne mais celui par le GPU ne fonctionne pas.

4.6. Résultats

La première partie décrit les résultats compilés dans notre affiche à la conférence *Quantum Technique in Machine Learning*. Ceux-si sont sur l'ensemble de données *mnist*. D'autres résultats partiels sur l'ensemble de données *Higg* ont été présentés à la conférence, mais comme ils n'ont pas été complétés par manque de temps de calcul, nous les avons omis.

Tableau 4.2. Résultats préliminaires présentés à QTML2019 [17].

Mnist validation	purement quantique	hybride linéaire	hybride
majorité	0.27	0.85	0.98
max	0.90	0.89	0.98

Le rouge pour la partie hybride linéaire signifie que nous n'avons pas corrigé les erreurs numériques par une factorisation QR des matrices unitaires de notre circuit. Par manque de temps, nous avons corrigé seulement les deux cas principaux : la méthode hybride avec un réseau neuronal et la méthode avec un circuit quantique seulement. La métrique de majorité est la plus sensible aux erreurs numériques. En général, après une correction la métrique max change peu. Nous expliquons la performance inférieure de la méthode hybride linéaire versus le circuit quantique simple par un sous-entrainement. Les circuits sont de taille 18 qubits avec des portes de 9 qubits avec topologie en quinconce et une profondeur de 13 niveaux.

La table 4.2 indique la performance sur l'ensemble de validation seulement afin de ne pas biaiser nos résultats ultérieurs sur l'ensemble de test. La table 4.3 présente les résultats finaux pour lesquels la meilleure époque est choisie par la performance du modèle sur la fonction de perte tandis que la table 4.4 présente les résultats finaux pour lesquels la meilleure époque est choisie par la performance du modèle sur la métrique max. Bien que nous préférerons une meilleure performance sur la métrique max, la fonction de perte reste un estimateur statistique non-biaisé. Considérant nos résultats présentement obtenus, nous ne savons pas vraiment quel serait le meilleur choix entre les deux. Vu la faible différence, nous penchons sur la fonction de perte à cause de ses propriétés statistiques.

Tableau 4.3. Résultats choisis par fonction de perte.

Mnist validation/test	purement quantique	hybride linéaire	hybride
perte	0.9762/0.9730	0.4465/0.4501	0.38996/0.3920
majorité	0.2567/0.2511	0.8214/0.8190	0.8520/0.8508
max	0.8973/0.9046	0.9163/0.9043	0.9839/0.9827
époque	43	58	59

Tableau 4.4. Résultats choisis par métrique max.

Mnist validation/test	purement quantique	hybride linéaire	hybride
perte	0.9795/0.9762	0.4546/0.4585	0.4145/0.4175
majorité	0.2523/0.2472	0.8178/0.8154	0.8495/0.8512
max	0.8985/0.9054	0.9171/0.9063	0.9841/0.9820
époque	44	54	57

Les performances des résultats finaux sont similaires à celle des résultats précédents, bien qu'un changement d'hyperparamètre cause une sous-performance pour les circuits seuls. Les deux autres nous semblent sous-entrainés. À 60 époques, Ils prennent tous les trois deux journées et demie à trois jours d'entrainement. Nous avons préféré investir notre temps de calcul dans sur circuit purement quantique plus large.

Pour ce circuit nous utilisons 22 qubits de largeur avec une topologie en quinconce de portes de 11 qubits et une profondeur de 13. Nous donnons deux copies de l'encodage en

Tableau 4.5. Résultats de 22 qubits purement quantiques.

Mnist	entrainement	validation	test
perte	0.6413	0.8777	0.8739
majorité	0.6159	0.3668	0.3692
max	0.8976	0.9104	0.9138
époque	2	2	2

amplitude de l'état sur dix qubits en plus de deux qubits ancillaires. Il va sans dire que cela prend du temps : environ deux jours par époque. De l'optimisation peut être faite, car les *batchs* de validation ont été exécutés sur le CPU au lieu du GPU. En effet, lorsqu'elles sont trop petites, *tensorflow* les déplace sur le CPU au lieu du GPU . Si elles sont trop grandes, elles utilisent trop de mémoire sur notre GPU. Il faut donc juger de leur taille. Une époque d'entrainement prend environ une journée à une journée et demie. Pour ces résultats nous avons été quelque peu chanceux de les obtenir à l'époque 2. Nous avons continué l'entrainement après, sans meilleurs résultats jusqu'à 8 époques.

À 91% de précision, nos résultats sur 22 qubits sont très bons. On peut observer l'extensibilité des circuits et l'augmentation de sa performance en augmentant son nombre de qubits et en lui donnant deux copies de son entrée. Ironiquement, même si le circuit de 22 qubits prend plus de temps à entrainer, sa convergence plus rapide résulte en de meilleures performances en moins de temps de calcul. Il est possible qu'ajouter l'étape optionnelle de l'algorithme 6 au momentum de Nesterov aide.

Chapitre 5

Extension et travaux futurs

Dans ce chapitre, nous décrivons les nombreuses extensions et nouvelles approches que nous voulons apporter au projet. La plupart d'entre elles n'ont pas été implémentées. Les principes de génie logiciel ont été la cause de leur non-implémentation. Le projet ayant une date butoir, nous ne pouvions pas continuer d'implémenter ces fonctionnalités logicielles à l'infini. Certaines d'entre elles nécessitent autant de travail que nous avons déjà mis sur ce mémoire.

5.1. Processeur quantique existant

La première idée venant à l'esprit est d'utiliser les ordinateurs quantiques existants pour exécuter nos circuits tel qu'appris. Comme nos circuits sont beaucoup plus profonds que les circuits couramment utilisés sur ceux-ci, compiler le circuit vers un processeur quantique contemporains nous semble voué à l'échec. Nous suggérons donc une approche différente. Nous entrainerons un circuit avec une topologie adaptée à l'ordinateur quantique physique avec lequel nous voulons exécuter le circuit. Naturellement, sa capacité sera limitée, car l'architecture sera beaucoup plus spécialisée. Il reste un moyen de s'en sortir en analysant les capacités de différentes topologies.

Par la suite, nous pouvons utiliser la mise à jour des circuits variationnels, pour corriger les erreurs intrinsèques du processeur quantique. Il est certain que cette correction d'erreur est inefficace comparée à un code correcteur quantique, mais ceux-ci ne s'appliquent pas sur les processeurs quantiques contemporains, leurs taux d'erreur étant trop élevés. Nous

pouvons voir cette façon de faire comme une initialisation du circuit variationnel, car la mise à jour ne fera pas seulement de la correction d'erreurs, mais aussi de l'apprentissage.

Cette procédure faciliterait l'implémentation de circuit variationnel, en échangeant des calculs quantiques coûteux pour du calcul classique beaucoup moins coûteux.

Des chercheurs de Tokyo [33] ont développé très récemment en parallèle une version de notre méthode et l'ont utilisée pour entrainer de petits circuits quantiques implémentés sur des processeurs physiques. Leurs tests en classification ont été réalisés sur des ensembles synthétiques : *Two Moons* dont la tâche est de classifier des points provenant de deux croissants de lune et *Circles*, l'équivalent pour les cercles. Ils ont aussi testé avec de la régression sur des fonctions synthétiques. Leurs circuits sont sur 4 qubits de large et ont une profondeur variant de 4, 7 et 10 qubits. Leurs ensembles de données pour la classification sont beaucoup plus « jouets » que notre ensemble *mnist*, mais considérant la taille de leurs circuits, ils sont parmi les seuls pouvant être utilisés. Nous pensons que l'ensemble *Iris*, vu sa faible taille, pourrait aussi être entrainé sur leurs circuits.

5.2. Paramétrisation efficace des portes quantiques

L'article *Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNNs* [16] propose une modification de la paramétrisation des matrices de l'article servant d'inspiration de notre algorithme, *Full-Capacity Unitary Recurrent Neural Networks* [36], basée sur des interféromètres quantiques de *An Optimal Design for Universal Multiport Interferometers* [10]. Comme leur paramétrisation donne des gains empiriques en temps de calcul, nous envisageons évidemment de l'essayer pour accélérer notre algorithme. De plus, cette paramétrisation permet de contrôler la capacité de la matrice unitaire paramétrée, pour laquelle on peut alors limiter l'espace de recherche. Ceci peut avoir des avantages, car il est possible que la solution ne se trouve pas dans cet espace. Dans notre cas, en plus des gains à l'apprentissage classique, comme elle est inspirée de matériel quantique, nous pensons pouvoir l'utiliser pour faciliter l'exécution du circuit appris sur un processeur quantique.

5.3. Exemples antagonistes

Dans la méthode hybride, les encodages proposés pourraient aider face aux exemples antagonistes (ceux qui seraient créés malicieusement par un adversaire). Puisque le circuit

quantique agit comme un *POVM*, il force le réseau classiques, pour obtenir une classification parfaite, à séparer les classes sur des vecteurs orthogonaux. Bien qu'une classification parfaite soit irréaliste en pratique, nous nous attendons à ce que le réseau classique sépare les classes le mieux possible sur des vecteurs orthogonaux pour obtenir une bonne classification. Notre hypothèse est que cette séparation réduirait ou éliminerait les exemples induits par le *softmax* tel que décrit dans *How the Softmax Output is Misleading for Evaluating the Strength of Adversarial Examples* [21]. Il est possible alors que ce détour par le quantique, même s'il n'est plus nécessaire pour évaluer le circuit rapidement, aide le réseau classique à être résistant aux exemples antagonistes.

Pour les exemples non induits par le *softmax* nous pensons, qu'au lieu d'imiter un *softmax* à la sortie du circuit, il serait préférable d'utiliser les espaces et qubits additionnels pour encoder une distribution de Dirichlet et permettre au modèle non seulement de répondre « je ne sais pas », mais aussi de quantifier son incertitude. Le modèle imiterait alors la sortie crée pour les réseaux classiques par Murat Sensoy, Lance Kaplan et Melih Kandemir dans *Evidential Deep Learning to Quantify Classification Uncertainty* [27]. Les auteurs de cet article montrent comment l'incertitude obtient de la robustesse face aux exemples antagonistes. Au lieu, de forcer le modèle à répondre une classe de nos données, il peut répondre « je ne sais pas », ce qui est avantageux lorsque confronté à de nouvelles données provenant d'une distribution différente. Par exemple, si une fois entrainé sur les données de *mnist* on le testait sur les données d'iris, au lieu d'avoir à répondre quel chiffre est le plus proche d'une fleur, il peut simplement répondre « je ne sais pas », réduisant le potentiel d'être mystifié par les exemples d'iris.

5.4. Boosting quantique

Nous voudrions implémenter les méthodes de *boosting* et *bagging* à l'aide de bifurcations (*forks*) quantiques développées dans *Parallel quantum trajectories via forking for sampling without redundancy* [22]. L'algorithme du *forking* quantique permet de faire une moyenne pondérée des sorties de plusieurs circuits. En entrainant de concert plusieurs circuits différents, nous obtenons du parallélisme de modèle, ce qui nous donne une superbe accélération pour du calcul distribué où chaque circuit aurait son propre GPU. Cette approche réduit de beaucoup les difficultés d'entrainement pour des circuits de plus grande quantité de qubits,

car elle nécessite peu de synchronisation favorisant ainsi les accès mémoires locaux et donc diminuant de beaucoup la latence liée à la mémoire qui serait nécessaire pour simuler un circuit de plus grande taille. Par contre, nous n'obtenons pas la pleine capacité des qubits additionnels, ayant fixé une partie de leur circuit. Malgré cet inconvénient, nous croyons que les bifurcations permettraient à notre modèle de ne plus devenir totalement obsolète suite à l'apparition des ordinateurs quantiques à grande échelle. Ceux-ci pourraient alors exécuter un ensemble de classificateurs entrainés par notre modèle à l'aide du *forking* quantique.

5.5. Génération de portes

Dans le modèle hybride, plutôt que générer la description d'un état quantique, nous pourrions générer les portes créant cet état. C'est-à-dire que pour une topologie donnée, le réseau classique pourrait modifier les paramètres des portes quantiques. Nous ne sommes pas limités seulement aux portes du début : le réseau classique à la possibilité de modifier n'importe quels ensembles de portes du circuit. Il est certain qu'il faut faire attention, car nous devons créer un nouveau circuit pour chacune des exécutions sur l'ordinateur quantique. Nous risquons donc de perdre des gains pratiques en temps de calcul. C'est pour cela que les paramètres devraient être très proches de ceux de la machine quantique, ce qui n'est pas irréaliste puisque plusieurs processeurs quantiques existants offrent déjà quelques portes paramétrées. Malheureusement, les portes paramétrées offertes sont en général de un qubit seulement. En prenant *pyquil* [29] à tire d'exemple, même s'il offre des portes *swap* et de changement de phase paramétrée contrôlé, leur modification est un changement de porte de un qubit du circuit qui les génère. Par contre, *pyquil* possède déjà un mode de compilation paramétrique permettant l'interface entre un modèle classique et les paramètres d'une porte.

5.6. Apprentissage par réduction

L'idée derrière cette approche est qu'il est difficile d'apprendre quantiquement. Par apprentissage quantique, nous entendons une mise à jour quantique à la manière des circuits variationnels. Ceci est à l'opposé de la procédure développée dans ce mémoire où nous faisons l'apprentissage classique d'un circuit quantique. D'où l'idée de l'oracle. En complexité du calcul, nous utilisons les oracles pour définir de nouvelles classes. Par exemple, si nous donnons un problème NP-complet en oracle à une machine de Turing, nous pouvons résoudre

les problèmes NP en faisant leurs réductions polynomiales vers celui-ci. Cette idée n'est pas pragmatique, car si une machine de Turing est capable d'implémenter efficacement l'oracle, il devient caduc.

Là où le quantique entre en jeux, c'est que contrairement aux classes de complexité, il existe des cas pratiques où une machine de Turing quantique peut résoudre des problèmes plus efficacement qu'on ne sait le faire classiquement. Donc, en donnant un problème pour lequel nous avons une implémentation efficace quantique, en présence d'un oracle nous avons simplement à apprendre la machine de Turing classique faisant la réduction à ce problème. Ceci devrait, en ignorant les problèmes liés à l'uniformité ou la non-uniformité des classes de complexité, nous donner la puissance de calcul du quantique dans lequel le problème donné en oracle est complet sans nécessiter de faire un apprentissage quantique. Nous apprenons seulement la partie classique.

Une autre façon de voir cette approche est que la partie difficile de l'apprentissage quantique, surtout pour les circuits variationnels, n'est pas l'apprentissage du circuit résolvant le problème, mais l'encodage de l'entrée pour celui-ci. Il nous manque des structures de données quantiques. En classique, nous avons plusieurs façons de représenter et de structurer les données. En quantique nous sommes limités en options et avons à faire plusieurs compromis tout en respectant la borne de Holevo. Comme il est lié à la borne de Holevo, cet encodage dépend de l'information que nous voulons obtenir. Nous proposons donc dans cette approche d'apprendre et même de surapprendre un encodage lié à une tâche d'apprentissage automatique. Nous fixons un problème pour lequel nous connaissons déjà un avantage quantique et apprenons un encodage, c'est-à-dire la réduction vers ce problème.

C'est un peu ce que fait l'algorithme de Shor en réduisant le problème de factoriser les entiers à celui de trouver la période de fonctions périodiques (injectives sur leur période) *et* en donnant un algorithme quantique efficace pour cette seconde tâche. Comme notre encodage est basé sur le problème d'apprentissage automatique, nous apprenons un encodage lié à celui-ci. Cela donne dans le surapprentissage, car il ne sera pas générique. Ceci est un compromis que nous pouvons accepter dans la quête de performance.

De préférence, la classe de complexité du problème quantique devrait être BQP, la classe de problèmes qui peuvent être résolus pas des algorithmes quantiques efficaces. Toutefois,

comme il existe des problèmes de BQP non complets pour lesquels aucune solution classique efficace n'a été trouvée (par exemple, la factorisation et le logarithme discret qui sont résolubles par l'algorithme de Shor), il ne semble pas nécessaire d'être strict sur la complétude pour obtenir des gains en pratique. À première vue, pour faciliter les implémentations, l'algorithme de Shor nous apparait comme le plus simple à utiliser contrairement aux problèmes BQP-complet, car ceux que nous avons vu sont des problèmes plus de physique que d'informatique.

5.6.1. Par renforcement

En termes d'implémentation, la première méthode de réduction venant à l'esprit est de considérer le circuit quantique fixe comme un environnement avec lequel le modèle classique apprend à interagir par renforcement. Nous trouverons que c'est déplacer les difficultés de la partie quantique à la partie classique. Mais a priori, cela nous semble réalisable bien que très difficile. Cependant il serait dommage d'être limité à de l'apprentissage par renforcement. Bien qu'il soit capable de certaines prouesses, nous n'obtenons pas la pleine puissance de l'apprentissage machine sans la possibilité d'apprentissage supervisé ou non supervisé.

5.6.2. Supervisée et non supervisée

Dans un monde idéal pour cette implémentation, nous désirerions que, en plus du fait que le problème du circuit soit vérifiable classiquement et donc dans NP, obtenir les gradients de la sortie du circuit par rapport à son entrée soit vérifiable classiquement et donc dans NP également. Les deux devraient en plus être dans BQP pour être efficaces quantiquement. Notre objectif dans ce cas est d'entrainer la partie classique en rétropropagation utilisant l'ordinateur quantique en oracle autant pour l'exécution du circuit que pour le calcul de sa dérivée. C'est un scénario très idéal et il se peut qu'il soit irréalisable. Par contre, comme il existe déjà des algorithmes de descente de gradient quantiques compatibles avec les modèles hybrides, il nous semble réaliste d'en trouver un qui fonctionnerait même s'il n'est pas aussi efficace qu'on pourrait en rêver.

Chapitre 6

Conclusion

Pour ce mémoire, nous voulions originellement évaluer empiriquement l'apprentissage machine quantique. Cet objectif ambitieux, considérant les contraintes imposées par la technologie moderne, ne pouvait qu'être partiellement réalisé, et ce sur l'apprentissage machine quantique à court terme.

À la fin de ce projet, nous trouvons que la première partie fut accomplie avec succès, car nous avons réussi à développer et implémenter une méthode pour évaluer empiriquement la capacité de petits circuits quantiques : un algorithme d'apprentissage de circuits quantiques par descente de gradient classique, tel que l'indique le titre de notre mémoire. Nous savions déjà que les contraintes rendraient impossible d'évaluer empiriquement plus que cela.

Pour la deuxième partie, les résultats obtenus sont loin d'être exhaustifs. Afin d'éviter les répétitions, le chapitre précédent passe sur une bonne quantité de méthodes à explorer pour continuer notre évaluation. D'autres ensembles de données que *mnist* et iris devraient aussi être évalués. À propos des résultats, pour le modèle purement quantique, bien que nous les trouvons très bons, ils sont loin d'être en compétition avec l'état de l'art des méthodes classiques, tandis que le modèle hybride ne semble pour l'instant aucunement tirer avantage du quantique.

Au final et c'est notre conclusion et la conclusion attendue : il reste beaucoup de travail à faire ! C'est pour cela que nous avons passé le chapitre 5 à discuter d'améliorations et de nouvelles approches. Même si nous avons obtenu de bons résultats, ce n'est pas le temps de chômer !

Bibliographie

[1] S. Aaronson. “Quantum Machine Learning Algorithms : Read the Fine Print”. In : *Nature Physics* 11 (avr. 2015), p. 291-293. doi : 10.1038/nphys3272.

[2] E. Aïmeur, G. Brassard et S. Gambs. “Quantum Speed-up for Unsupervised Learning”. In : *Machine Learning* 90.2 (fév. 2013), p. 261-287. issn : 1573-0565. doi : 10.1007/s10994-012-5316-5.

[3] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy et R. Melko. “Quantum Boltzmann Machine”. In : *Phys. Rev. X* 8 (2 mai 2018), p. 021050. doi : 10.1103/PhysRevX.8.021050. eprint : arXiv:1601.02036. url : https://link.aps.org/doi/10.1103/PhysRevX.8.021050.

[4] M. Arjovsky, A. Shah et Y. Bengio. “Unitary Evolution Recurrent Neural Networks”. In : *Proceedings of the 33rd International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA : JMLR.org, 2016, p. 1120-1128. arXiv : 1511.06464.

[5] M. Benedetti, E. Lloyd, S. Sack et M. Fiorentini. “Parameterized Quantum Circuits as Machine Learning Models”. In : *Quantum Science and Technology* 4.4 (nov. 2019), p. 043001. doi : 10.1088/2058-9565/ab4eb5. arXiv : 1906.07682. url : https://doi.org/10.1088%2F2058-9565%2Fab4eb5.

[6] Y. Bengio, N. Boulanger-Lewandowski et R. Pascanu. “Advances in Optimizing Recurrent Networks”. In : *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), p. 8624-8628. arXiv : 1212.0901.

[7] C. H. Bennett, E. Bernstein, G. Brassard et U. Vazirani. “Strengths and Weaknesses of Quantum Computing”. In : *SIAM Journal on Computing* 26.5 (1997), p. 1510-1523. doi : 10.1137/S0097539796300933.

[8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe et S. Lloyd. "Quantum Machine Learning". In : *Nature* 549.7671 (sept. 2017), p. 195-202. issn : 1476-4687. doi : 10.1038/nature23474. arXiv : 1611.09347.

[9] W. M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. 2e éd. Pure Appl. Math. Orlando, FL : Academic Press, 1986. url : https://cds.cern.ch/record/107707.

[10] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer et I. A. Walmsley. "An Optimal Design for Universal Multiport Interferometers". In : *Optica* 3.12 (déc. 2016), p. 1460-1465. doi : 10.1364/OPTICA.3.001460. arXiv : 1603.08788. url : http://www.osapublishing.org/optica/abstract.cfm?URI=optica-3-12-1460.

[11] C. M. Dawson et M. A. Nielsen. "The Solovay-Kitaev Algorithm". In : *Quantum Info. Comput.* 6.1 (jan. 2006), p. 81-95. issn : 1533-7146. arXiv : quant-ph/0505030.

[12] I. Goodfellow, Y. Bengio et A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[13] E. Grant, M. Benedetti, S. Cao, A. Hallam, J. Lockhart, V. Stojevic, A. G. Green et S. Severini. "Hierarchical Quantum Classifiers". In : *npj Quantum Information* 4 (2018), p. 1-8. doi : 10.1038/s41534-018-0116-9. arXiv : 1804.03680.

[14] A. W. Harrow, A. Hassidim et S. Lloyd. "Quantum Algorithm for Linear Systems of Equations". In : *Phys. Rev. Lett.* 103 (15 oct. 2009), p. 150502. doi : 10.1103/PhysRevLett.103.150502. arXiv : 0811.3171.

[15] A. S. Holevo. "Bounds for the Quantity of Information Transmitted by a Quantum Communication Channel". In : *Problemy Peredachi Informatsii* 9.3 (1973), p. 3-11.

[16] L. Jing, Y. Shen, T. Dubček, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark et M. Soljačić. "Tunable Efficient Unitary Neural Networks (EUNN) and Their Application to RNNs". In : *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia : JMLR.org, 2017, p. 1733-1741. arXiv : 1612.05231.

[17] A. Lamarre. "Learning Quantum Circuits with Classical Gradient Descent". In : *Proceedings of the 3th Quantum Techniques in Machine Learning Conference*. 2019, p. 150-152.

[18] S. Lloyd, M. Mohseni et P. Rebentrost. "Quantum Principal Component Analysis". In : *Nature Physics* 10.9 (sept. 2014), p. 631-633. ISSN : 1745-2481. DOI : 10.1038/nphys3029. arXiv : 1307.0401. URL : https://doi.org/10.1038/nphys3029.

[19] T. M. Mitchell. *Machine Learning*. 1re éd. New York, NY, USA : McGraw-Hill, Inc., 1997. ISBN : 0070428077, 9780070428072.

[20] M. A. Nielsen et I. L. Chuang. *Quantum Computation and Quantum Information : 10th Anniversary Edition*. New York, NY, USA : Cambridge University Press, 2011. ISBN : 1107002176, 9781107002173.

[21] U. Ozbulak, W. D. Neve et A. V. Messem. "How the Softmax Output Is Misleading for Evaluating the Strength of Adversarial Examples". In : (2018). arXiv : 1811.08577.

[22] D. K. Park, I. Sinayskiy, M. Fingerhuth, F. Petruccione et J.-K. K. Rhee. "Parallel Quantum Trajectories via Forking for Sampling without Redundancy". In : *New Journal of Physics* 21.8 (août 2019), p. 083024. DOI : 10.1088/1367-2630/ab35fb. arXiv : 1902.07959.

[23] P. Rebentrost, M. Mohseni et S. Lloyd. "Quantum Support Vector Machine for Big Data Classification". In : *Phys. Rev. Lett.* 113 (13 sept. 2014), p. 130503. DOI : 10.1103/PhysRevLett.113.130503. arXiv : 1307.0471.

[24] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac et N. Killoran. "Evaluating Analytic Gradients on Quantum Hardware". In : *Phys. Rev. A* 99 (3 mar. 2019), p. 032331. DOI : 10.1103/PhysRevA.99.032331. arXiv : 1811.11184.

[25] M. Schuld, A. Bocharov, K. M. Svore et N. Wiebe. "Circuit-Centric Quantum Classifiers". In : *Phys. Rev. A* 101 (3 mar. 2020), p. 032308. DOI : 10.1103/PhysRevA.101.032308. arXiv : 1804.00633. URL : https://link.aps.org/doi/10.1103/PhysRevA.101.032308.

[26] M. Schuld, I. Sinayskiy et F. Petruccione. "An Introduction to Quantum Machine Learning". In : *Contemporary Physics* 56.2 (2015), p. 172-185. DOI : 10.1080/00107514.2014.964942. arXiv : 1409.3097.

[27] M. Sensoy, L. Kaplan et M. Kandemir. "Evidential Deep Learning to Quantify Classification Uncertainty". In : *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, p. 3179-3189. arXiv : 1806.01768.

[28] J. W. Siegel. “Accelerated Optimization with Orthogonality Constraints”. In : (2019). eprint : arXiv:1903.05204.

[29] R. S. Smith, M. J. Curtis et W. J. Zeng. “A Practical Quantum Instruction Set Architecture”. In : (2016). arXiv : 1608.03355 [quant-ph].

[30] H. D. Tagare. *Notes on Optimization on Stiefel Manifold*. Rapp. tech. Yale University, 2011.

[31] G. Verdon, J. Pye et M. Broughton. “A Universal Training Algorithm for Quantum Deep Learning”. In : (2018). arXiv : 1806.09729.

[32] K. H. Wan, O. Dahlsten, H. Kristjánsson, R. Gardner et M. S. Kim. “Quantum Generalisation of Feedforward Neural Networks”. In : *npj Quantum Information* 3.1 (sept. 2017), p. 36. issn : 2056-6387. doi : 10.1038/s41534-017-0032-4. arXiv : 1612.01045. url : https://doi.org/10.1038/s41534-017-0032-4.

[33] M. Watabe, K. Shiba, M. Sogabe, K. Sakamoto et T. Sogabe. “Quantum Circuit Parameters Learning with Gradient Descent Using Backpropagation”. In : (2019). arXiv : 1910.14266.

[34] N. Wiebe et L. Wossnig. “Generative Training of Quantum Boltzmann Machines with Hidden Units”. In : (2019). arXiv : 1905.09902.

[35] M. M. Wilde. *Quantum Information Theory*. 1re éd. New York, NY, USA : Cambridge University Press, 2013. isbn : 1107034256, 9781107034259.

[36] S. Wisdom, T. Powers, J. R. Hershey, J. L. Roux et L. Atlas. “Full-Capacity Unitary Recurrent Neural Networks”. In : *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. Barcelona, Spain : Curran Associates Inc., 2016, p. 4887-4895. isbn : 9781510838819. arXiv : 1611.00035.